SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

RESEARCH INTO SOFTWARE EXECUTIVES FOR SPACE OPERATIONS SUPPORT

FINAL REPORT

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Prepared by:
Mark D. Collier

Prepared for:
NASA
Johnson Space Center
Houston TX 77058

September 28, 1990

Approved:

Melvin A. Schrader

Melvin A. Schrader, Director
Data Systems Department

1.0 Introduction

This document serves as the final report for the activity on NASA Grant NAG 9-340, which is entitled "Research into Software Executives for Space Operations Support". The purpose of this grant is to research concepts pertaining to a software (workstation) executive which will support a distributed processing command and control system characterized by high-performance graphics workstations used as computing nodes.

2.0 Research Background

During the past few years and increasing in the future, many centralized computing installations are migrating to environments characterized by distributed processing. This migration is driven primarily by the low cost and high performance delivered by state-of-the-art graphic workstations. Such an environment normally consists of a large number of workstations which are in turn connected via one or more high-speed networks.

Although a workstation-based distributed processing environment offers many advantages, it also introduces a number of new concerns. One problem is that engineering-class workstations most commonly use the UNIX operating system, which is difficult for computer novices to use effectively. Also, connecting a large number of workstations and expecting them to work as an integrated system is not easily achieved. The introduction of so many separate processors makes configuration management and security a real concern. In fact, the very flexibility which is inherent in workstations often becomes a problem. This is especially true for real-time critical command and control systems in which a failure or security break could have disastrous results.

In order to solve these problems, allow the environment to function as an integrated system, and present a functional development environment to application programmers, it is necessary to develop an additional layer of software. This "executive" software integrates the system, provides real-time capabilities, and provides the tools necessary to support the application requirements. Such an executive will be required for use in evolving systems such as the ground-based control centers planned at Johnson Space Center. These command and control environments will use a distributed processing architecture to provide real-time processing of telemetry and command data.

3.0 Research Results

All research performed was geared toward identification and introduction of software technology into a workstation executive. The research results are represented in the following efforts and reports:

- Executive Survey and Requirements Report - This report represents the primary effort of this grant. This effort included a survey of NASA and commercial sites for systems relevant to workstation executives, a survey of current standard system software technology, and generation of a set of requirements for a Concept Workstation Executive.
- Graphics Update Report - SwRI performed an analysis of existing graphic user interface and drawing software applicable to workstation executive requirements. This report identifies software such as X Windows, Motif, GKS, and PHIGS and in which applications this software should be used.
- Analysis of MCCU Workstation Executive Design Report- to provide background into the direction of a typical executive, SwRI analyzed and commented on the preliminary and critical design of the MCCU Workstation Executive.

- Analysis of POSIX 1003.4 and 1003.6 Report - in order to determine if the proposed POSIX standards for real-time and security extensions supported the requirements of workstation executives, SwRI analyzed and commented on the current drafts of the 1003.4 (real-time) and 1003.6 (security) standards.
- Inter-language Support Report - SwRI performed a brief survey of current efforts to define a standard interface between the C, Ada, and FORTRAN high-level languages.
- Load Sharing Prototype - To demonstrate proof-of-concept of a capability specified in the Concept Executive, a load sharing prototype was developed. This prototype records the load of each workstation in a set of workstations and for each job request, schedules the job on the workstation with the highest amount of available processing capability.

1.0 Introduction

During the past few years, many computing environments have migrated from centralized processing architectures to ones characterized by distributed processing. While distributed processing systems have existed for years, the recent increase is driven by the availability of high-performance engineering class workstations. An engineering class workstation is defined as having a high-performance central processing unit (CPU), a floating point accelerator (FPA), at least 256 colors, at least 1 mega-pixel resolution, and based on the UNIX operating system. Use of UNIX is crucial as it normally carries with it network and graphics support.

A distributed processing system based on such workstations normally consists of one or more high speed networks which provide connectivity for the workstations. A distributed processing architecture of this type offers the following advantages:

- High performance - the continuing development of high performance microprocessors allows workstations to provide high integer, floating point, and graphics performance. Many workstations also provide support for multiple processors.
- Flexibility - the UNIX operating system, graphics, and networking software provide a flexible and powerful development environment which supports a wide variety of application requirements.
- Autonomy - users have a high level of control over a powerful local computing resource and are marginally affected by overall system load.
- Expandability - workstations may be added to the network to support new users or increased processing requirements.
- Interoperability - by utilizing standards for application development, workstations from many different vendors may be utilized in the same network. An obsolete workstation may be replaced with a new workstation which provides greatly improved performance. The new workstation may be from the same or a different vendor.

Perhaps the most exciting advantage of using workstations is the availability of hardware and software standards. The development of standards promises interoperability among many different workstation vendors. Workstation technology is among the fastest (if not the fastest) growing in the industry and there is intense competition for market share. This competition is forcing development of systems which often double in performance every year. By basing application software on standards, an installation can take advantage of this competition and procure the system which best suits their needs. In the future, as new workstations are introduced, they can be purchased and installed without a major reinvestment in software porting or redevelopment.

1.1 Real-time Distributed Processing Systems

For several years, system developers have been using a distributed processing architecture to support real-time systems. A real-time system is one in which both the time of delivery and content of data is significant in determining the validity of the information. A widely accepted view is that a distributed processing architecture is superior for real-time systems, as a large and complex task may be partitioned amongst a number of separate processors, each of which is responsible for a specific function or set of functions. This type of architecture offers the following advantages:

- Deterministic response - by devoting a processor to a given function, it is possible to insure the ability of the processor to respond to and process events in real-time.

- Fault tolerance - distributed processing by its very nature is fault tolerant for if one processor fails, only the associated functionality is lost (in a well designed system). Fault tolerance can be improved by using redundant processors and/or a migration scheme in which functions are moved from a failed processor to another.
- Expandability - in a well-defined system, as processing requirements increase, existing processors may be upgraded or new processors may be added.

Using a distributed architecture for a real-time system also introduces a number of disadvantages, including:

- Data and control distribution - in order to process data, the data must be distributed to all processors. Data distribution must be performed in real-time, as a significant delay will offset the advantages of distributed processing.
- Configuration management and security - the use of many separate processors, each with its own operating system and application software, makes it difficult to insure consistency across the configuration. This concern is especially apparent for critical command and control systems in which a failure or security break could have disastrous results.

Real-time systems utilize a wide variety of processors and networks. This varies from completely custom and proprietary systems to modified Commercial-Off-The-Shelf (COTS) hardware and software to proprietary COTS systems. Although in most instances it is possible to expand such systems, additional resources have to be obtained through the original developer at no small cost. In such instances, system users are not able to procure COTS components in order to expand or improve the system. Due to the high cost of expansion, performance increase requirements are often ignored until such time that the system can not meet requirements and has to be replaced. This is unacceptable for static systems and especially so for dynamic systems which must constantly expand to address new requirements.

1.2 Workstations in the Real-time World

An obvious solution is to use workstations as the processors in a distributed real-time system. This type of architecture will allow the system to realize the advantages of a real-time system and offer the expandability and interoperability advantages of using workstations.

Unfortunately, using workstations in a real-time environment introduces several problems, including the following:

- Real-time response - standard UNIX is not a real-time operating system. At this time it is not possible to use a standard set of UNIX real-time extensions.
- Immature standards - workstation software standards are not yet fully mature. Standards exist, but there are new standards expected to be in place in the near future.
- Configuration management - connecting a number of workstations into a network and allowing them to work as an integrated system is not easily achieved. The very flexibility advantages of workstations causes configuration management problems.

One approach toward solving these problems, allowing the environment to function as an integrated system, and presenting a standardized development environment to application programmers, is to develop an additional layer of software.

1.3 The Workstation Executive

The workstation executive is a software subsystem which bridges the gap between the UNIX operating system and the specific requirements of an environment's application programmers. The workstation executive is not necessarily an additional layer of software, but rather a collection of COTS and custom software which presents an integrated, functionally complete, standardized, and real-time environment to application programmers. The three primary goals of the workstation executive are:

- Standardization - the workstation executive will include standard software and interfaces which insure the portability of both the executive and all applications.
- System integration and application support - the workstation executive will present an integrated interface which meets the requirements of the environment's application programmers.
- Environment control - the workstation executive will define the configuration, control access to resources, and insure the stability of the system during operations.

The purpose of this document is to present the results of the following research on workstation executives:

- Determine the definition of an executive.
- Briefly describe the system for which the executive is defined.
- Determine the scope of the executive within such an environment.
- Determine the requirements for a concept workstation executive.
- Present the results of the executive technology survey.

The end product of this document is a set of requirements which defines a Concept Executive. This Concept Executive will define a system for a real-time telemetry processing command and control environment. The use of the term "concept" implies that the requirements will be specified at a level which is independent of a specific environment. Once the requirements are complete, the Concept Executive may be applied to a specific environment, expanded for details, and used as the basis for system development.

1.4 Document Map

This document is divided into ten chapters and a bibliography. The chapters and topics are as follows:

- **[1]** Introduction - (current chapter) introduces the requirement for a software (workstation) executive in real-time spacecraft telemetry processing system characterized by distributed processing.
- **[2]** Research Background - describes the phases of this grant and previous research which is relevant to this effort.
- **[3]** Definition of an Executive - defines the basic function of a real-time software executive.
- **[4]** Target System Description - briefly describes system for which the executive is targeted.
- **[5]** Concept Executive Description - introduces the Concept Executive and its goals in the target environment.

- [6] Concept Executive Functional Requirements - presents the requirements which make up the Concept Executive.
- [7] Survey Introduction - an introduction to the technology survey which was conducted before and during the Concept Executive requirements definition..
- [8] Executive Systems - includes a summary of each executive reviewed.
- [9] User Interface Systems - includes a summary of each user interface reviewed.
- [10] Standards - includes a summary of each new standard reviewed.
- Appendix A - Bibliography.

It is suggested that the reader review chapters 1 through 6 for information on the Concept Executive. During the review process, the reader may refer to Chapters 7 through 10 for more information on systems which are specified or referenced by the Concept Executive requirements.

2.0 Research Background

This document presents the research findings of two phases of NASA Grant NAG 9-340, which is entitled "Research in Software Executives for Space Operations Support." A "Software Executive" is a term which covers a wide area of software functionality. This grant is intended to define the purpose of an executive and indicate the scope of its responsibility within a real-time spacecraft telemetry processing command and control environment. This grant includes the following three phases:

- Phase 1 - Survey NASA and commercial sites for executive systems and review current standards and software technology which is relevant to the research. The results of phase 1 are presented in chapters 7 through 10.
- Phase 2 - Define an "executive", determine its scope within a real-time spacecraft telemetry processing command and control environment, and develop a set of requirements for a concept executive which could be used as the basis for such an environment. The results of this phase are presented in chapters 2 through 6.
- Phase 3 - As proof of concept, apply the concept executive to an actual environment at NASA Johnson Space Center. This process will involve adding detailed requirements and prototyping the executive or critical portions of it.

This document summarizes the results of Phases 1 and 2. The results of phase 3 will be presented in a separate document which will be delivered at a later date.

2.1 The HISDE Prototype

For NASA Grant NAG 9-269, which was entitled "Research in Software for Space Operations Support", Southwest Research Institute developed the Hardware Independent Software Development Environment (HISDE) prototype to serve as proof-of-concept for a hardware-independent workstation executive. The HISDE prototype introduced a number of advanced software technologies and concepts including:

- Extensive use of widely accepted industry standards, including each of the following:
  - SVID UNIX operating system.
  - C programming language.
  - X Windows.
  - GKS and PHIGS.
  - ISO OSI communications.

  Through the use of standards, HISDE was easily ported across multiple vendor workstations.
- Open use of UNIX - through a more flexible design and configuration management scheme, HISDE provided access to the UNIX file system via the familiar UNIX command line interface.
- Configuration Manager (CM) Workstation - this concept involves a workstation to which all user applications are loaded with certified libraries prior to being mission certified and uploaded to a configuration management host.

The HISDE prototype is significant to the current research effort as many of the standards and concepts embodied by HISDE are retained in the concept executive. For more information on the HISDE prototype, refer to the Final Report for NASA Grant NAG 9-269.

3.0 Definition of an Executive

The first task of this document is to define the term "executive." This is necessary in order to determine the scope of functionality of the executive within the described environment. The term "executive" is somewhat generic and interpreted in various ways by different individuals. The most common definition of an executive is a low-level software subsystem which provides application programmers with a more convenient means of accessing the real-time computing capability of the hardware. As described in [BARA86], "a real-time executive is a software kernel which acts as an interface between the application software and the CPU hardware." The executive will normally utilize the available software functionality such as I/O processing, file management, and other primary functions and then provide the means whereby these functions are used in a real-time, process-oriented manner. As described in [FITZ87], these functions include "task scheduling, time/timer management, memory/buffer management, and interrupt handling." Similar functions are described in [WANG87] as "processor management, timer management, I/O management, and task work storage allocation."

An executive is normally used on a system which involves a custom hardware configuration designed for a particular application. Such a hardware system will often not provide a complete real-time multi-tasking operating system. Rather it will include a very basic set of I/O, file management, and hardware interface software functions (some of which may be custom developed as well). This functionality is then tied together by the executive. By providing functions such as process scheduling, the executive allows application programmers to execute and control a number of programs. Although similar in function, the executive is not the operating system. Rather as described in [CLAU87], "The executive is at the top level of the operating system. The executive is the central, coordinating component of the operating system."

The executive provides primitive functions to the applications programmer. As described in [FITZ87], "Primitives provided by the executive should satisfy the classic needs of real-time applications, at the same time having sufficient power and functionality to make applications development straightforward." This thought is shared in [CLAU87], as "services of the OS and executive must cover completely the needs of the applications. Services are just abstractions; they package the power of the hardware and make it easier to use by managing details irrelevant to the end user." The functions provided by the executive should be complete and it should not be necessary for the application programmer to develop additional functionality.

Although an executive is normally defined as a low-level software system, it is more important to concentrate on the purpose of the executive as a provider of services to the application programmer. The purpose of the executive is to attempt to bridge the gap between system hardware and software and the requirements of the application programmer. This is true whether the executive is a simple layer on top of device drivers or is a complete multi-tasking, real-time operating system. This discussion will focus on the instance of layering an executive on top of an operating system. An operating system will have been designed for general purpose usage in which a large variety of applications may be developed. As such it requires additional functionality to make it convenient to use by specific application programmers. This is the purpose of the executive, to tailor the operating system to be as convenient as possible for applications programmers.

When designing the executive, it is important to clearly define the requirements of the system and its application programmers (Note that the requirements of the "system", such as configuration management, may conflict with or modify the requirements of programmers and users). As the

system and its applications all depend on the executive, it must be efficient and well-suited for the environment. The executive must neither be too generic nor too specific. Being too generic will result in inefficiency; being too specific will require application programmers to bypass the convenience functions provided by the executive. As stated by [CLAU87], "If it is necessary for any reason to break into the package, the convenience vanishes."

Another function of the executive is that it must often hide or mask functionality provided by a general purpose operating system. The very flexibility of many operating systems often becomes a problem in environments which demand high security and a stable configuration. In such cases, the executive will either remove or buffer the functionality in question from users of the system.

In view of the information presented, the definition of an executive is **"A software subsystem which adds upon the inherent system hardware and software functionality to meet the requirements of the environment and its applications programmers."** Figure 3-1 illustrates the place of the executive in a system.

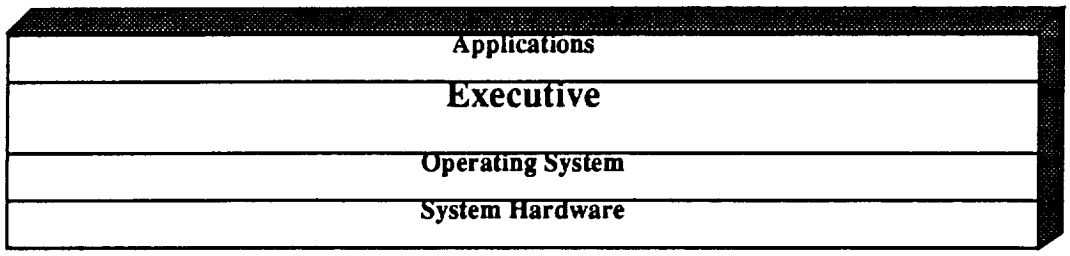


Figure 3-1 Executive in System Hierarchy

An important point is that the executive tailors functionality to support the requirements of application programmers. This is opposed to providing an interface directly usable by the end users of the system.

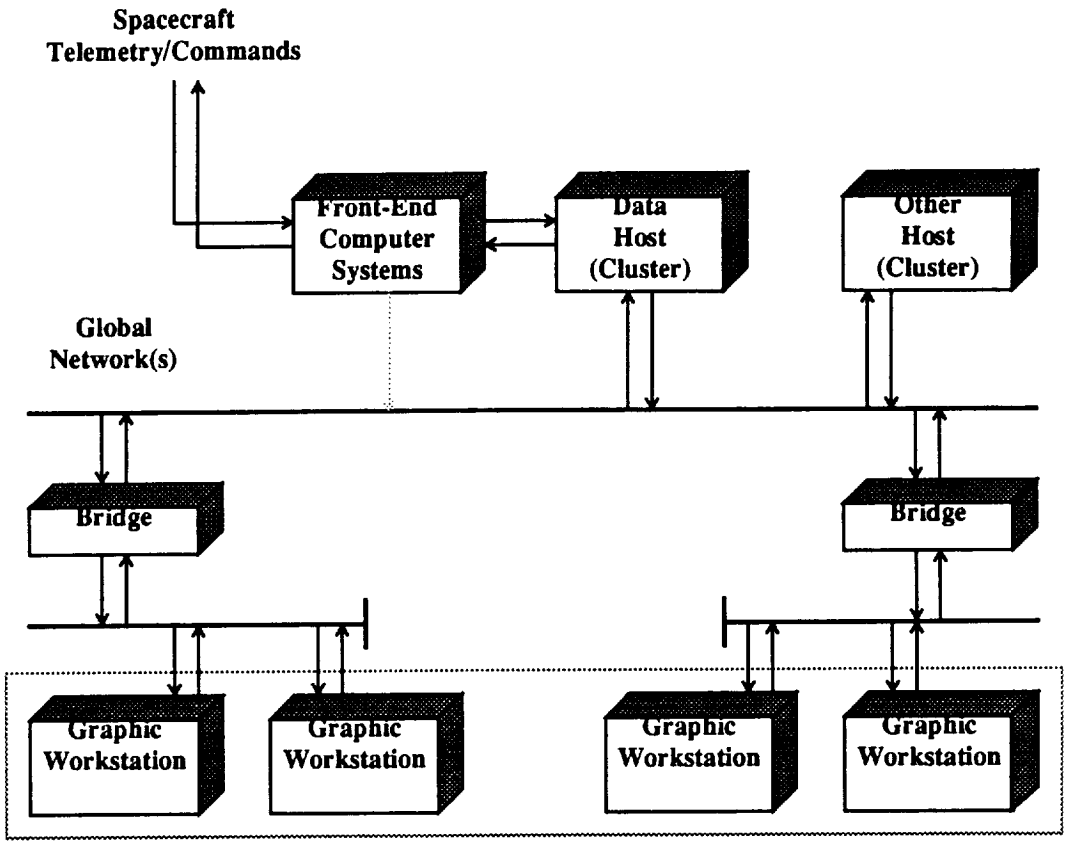
4.0 Target System Description

The end result of this document is a set of requirements which defines a concept executive which will manage a network of workstations in order to support a real-time spacecraft telemetry processing command and control environment. The particular type of environment is a ground-based control center responsible for monitoring and control of space vehicles. Rather than describing an actual executive implementation for such an environment, the concept describes requirements and software principles which may be used as the foundation for an implementation of this type. The concept focuses on applying state-of-the-art software technology which is generic and expandable in order to allow incorporation of new hardware and software technology.

To prevent the concept from being too generic and impossible to apply in the desired control center environments, it is necessary to describe the system and make certain assumptions. This control environment will be characterized by the following:

- Telemetry processing system - this will be a data-driven system in which the primary input is provided in the form of a cyclic block of real-time telemetry data. This telemetry data corresponds to status information sent from on-board spacecraft systems. The primary purpose of the system is to provide this information to end users to allow monitoring and control of on-board systems.
- Workstations - all data will be made available to and processed by individual workstations. These workstations will be the primary computing elements and will be capable of directly accessing, manipulating, and displaying telemetry data. Workstations will also be capable of initiating spacecraft command sequences.
- Networks - all workstations making up the environment will be directly or indirectly connected to one or more high-speed global networks.
- Real-time throughput of data - the ability to access, manipulate and compute, display, and generate command data must be available in real-time.
- Graphics - much of the presentation of data is performed via high-performance color graphics. Data will be displayed in forms of color-coded text, plots, and other forms of graphic presentation.
- Host support - the system will rely on one or more hosts to perform centralized functions such as data provision, data recording, software archival, network management, systems status, and other functions.
- Configuration management and security - it is assumed that the operational system will employ a rigid configuration management and security policy to insure software certification and control of user interaction during operations. It is assumed that a host will be the central repository of all certified software.

The primary hardware components and data flow of the system is illustrated in Figure 4-1.



Legend:

- *The dotted line indicates the extent of responsibility of the executive.*
- *The dotted lines connecting the Front-End Computer System and the Global Network(s) indicate that raw data may or may not be placed on the network.*

Figure 4-1 Target Environment Hardware Configuration

The executive concept will not directly describe or suggest the most appropriate hardware configuration. Rather, as described later in the requirements, it will provide a concept which will support different configurations and allow simple addition or replacement of hardware.

5.0 Concept Executive

A workstation executive software subsystem is required in the target environment to support integrated utilization of workstations, networks, graphics, and the UNIX operating system. The workstation executive will provide the following functions:

- Standardization - the workstation executive will include standard software and interfaces which insure the portability of both the executive and applications.
- System integration and application support - the workstation executive will present an integrated software interface which meets the requirements of the systems application programmers.
- Environment control - the workstation executive will define the configuration, control access to resources, and insure the stability of the system during operations.

This document describes a "concept" workstation executive which provides these basic services. The Concept Workstation Executive (or "Concept Executive") will establish its scope within the target environment and then describe a set of requirements. This description is a concept, as it specifies the requirements at a high-level and is intended to support multiple control center environments. Rather than focusing on environment specific details, the concept identifies and applies state-of-the-art standards and technologies. The concept also focuses on standards which provide a clean migration to more global standards to be defined in the near future. The primary goal of the Concept Executive is to provide the basis of a system which supports multiple environments and will have a life cycle extending far into the future.

5.1 Types of Programmers and Users

For this discussion, it is necessary to distinguish the terms "user" and "applications programmer." A user is the individual who is knowledgeable and responsible for an onboard or ground-based support system. This individual is able to examine data and determine the distinction between normal and abnormal operation. This individual is not however assumed to be knowledgeable about software development, the executive, or the operating system. This is not to say that individual users are not able to develop sophisticated applications, but rather that this is not the general case.

An applications programmer is an individual who takes the requirements generated by a user, and by using the features provided by the operating system and the Concept Executive, translates the requirements into an executable program. An applications programmer is assumed to be knowledgeable about the operating system, the Concept Executive, and software development in general.

The reason for this distinction is that the Concept Executive is intended to provide functionality to the level required by the application programmer. This makes the scope of the Concept Executive much more manageable and is reasonable as the executive must support development of a wide variety of applications. If the Concept Executive was designed to directly meet the requirements of end users, then the scope of the executive would increase dramatically. Addressing the requirements of non-programming users requires development of a sophisticated and integrated set of tools which can be directly used by the end user. Although this is a reasonable goal, the associated applications should build upon the interfaces provided by the Concept Executive, rather than being considered a part of the executive itself.

5.2 Concept Executive Scope

Figure 5-1 illustrates the different layers of software which will be present in the complete system.

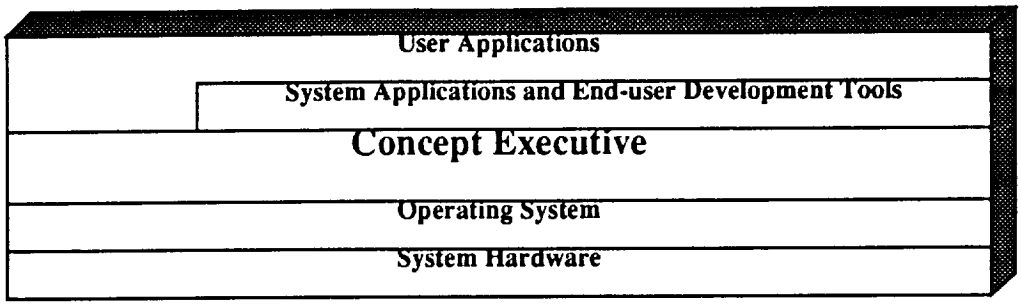


Figure 5-1 Concept Executive in System Hierarchy

All application software interfaces with the operating system through the Concept Executive. This is because the Concept Executive, although it allows access to much of the operating system functionality, actually determines what is and is not legal to use during different modes of operation. This is necessary for configuration management, system security and to insure development of standard applications.

Residing above the workstation executive are two distinct layers of application software, including:

- System Applications and End User Development Tools - this layer of software consists of system applications intended to simplify use of the system for application programmers and/or end users. This broad layer of software tailors the environment to the requirements of different types of users.
- User applications - this layer of software consists of applications which directly satisfy user requirements and are used for flight support. This layer includes applications developed directly upon the Concept Executive and with the aid of System Applications and End User Development Tools.

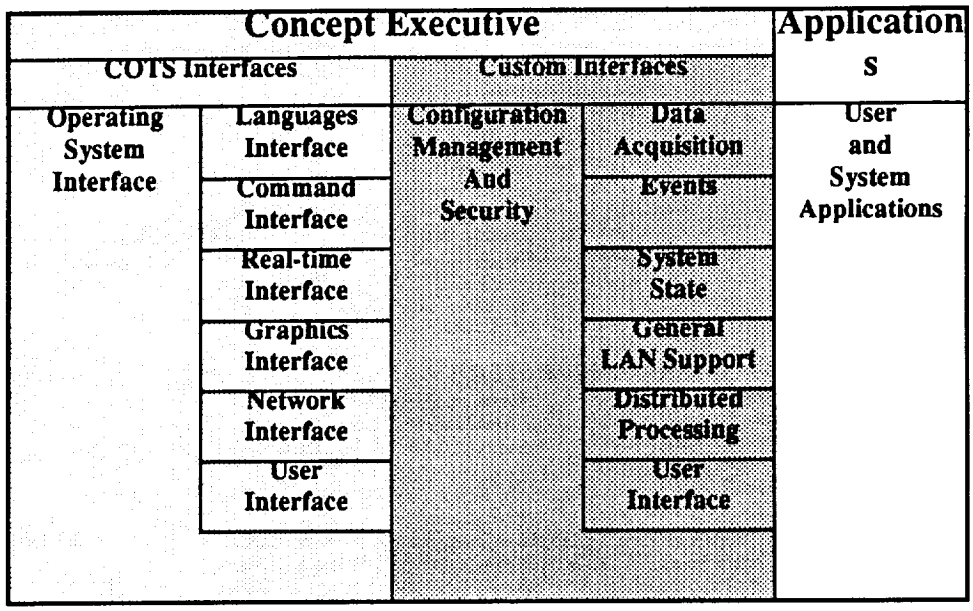
The separation between the Concept Executive and the System Applications is what primarily defines the executive scope. The location of this separation greatly affects what is and is not considered part of the executive. The Concept Executive is defined to provide an integrated set of services which supports the common requirements of system and application programmers. As such, the Concept Executive does not provide a diverse set of specialized interfaces which are tailored to specific requirements.

The Concept Executive primarily consists of COTS standards and a minimum of efficient custom software which is needed to support the environment. The Concept Executive tailors itself to the environment by providing support for networking, graphics, configuration management and other required primary interfaces. The Concept Executive also recognizes the primary functional requirement of access to operational data. Therefore the Concept Executive provides a basic set of interfaces which simplify and isolate this process. The Concept Executive assumes that additional higher level software subsystems will be developed in order to support more specific requirements.

5.3 Concept Executive Contents

The Concept Executive shall present a standardized development interface to the application programmers of the target environment. The Concept Executive will consist of (COTS) and custom software which cooperate to achieve this goal. Emphasis is of course placed on the use of COTS

software as this reduces costs and generally provides a more robust system. The basic contents of the Concept Executive are summarized in Figure 5-2.



Legend:

- *The operating system interface is extended to indicate access by custom interfaces.*
- *Configuration Management and Security is extended to indicate access by system and user applications.*

Figure 5-2 Concept Executive Contents

As described, the goals of the Concept Executive are standardization, system integration and application support, and environment control. Each goal and the subsystems which satisfy the related requirements are described in more detail in the following subsections.

5.3.1 Standardization

In order to achieve the expandability and interoperability advantages of using workstations, it is necessary to develop portable software. The Concept Executive will support development of portable applications by the following:

- Use of standards - the Concept Executive and the interface which it presents to application programmers will be standardized. The Concept Executive itself will be portable, with the caveat that the executive may use certain features to obtain real-time response. The interface presented to application programmers will be totally standard and will allow development of 100% portable applications.
- Function isolation - although based on standards, the Concept Executive must still provide a real-time environment. Due to this, certain non-standard functions may be required in order to achieve real-time performance. For such functions, the Concept Executive will present standard interfaces which isolate the non-standard functions. In this way, applications will be isolated from system-specific functions.

- Configuration independence - to be truely portable, the Concept Executive must support transparent operation in a variety of configurations. The Concept Executive makes no assumptions about the arrangement of the global networks or the workstations themselves. The Concept Executive will support several configurations which adequately meet the real-time requirements.
- Future standards directions - the drive for open systems is a relatively new concern for which complete standards are not yet available. The Concept Executive shall use interim standards with the goal in mind of migrating to final standards when actually available.

All of the standards upon which the Concept Executive is based are taken from or represent logical progressions from the standards specified by the HISDE prototype. Additional standards which had not been finalized during development of the HISDE prototype will be specified for the Concept Executive.

5.3.2 System Integration and Application Support

The second goal of the Concept Executive is to present an integrated and functional interface to the application programmers of the environment. This standard-based interface will support the entire spectrum of system and application requirements in the environment. The different areas of functional support include:

- Processing of flight data - the Concept Executive will provide real-time services which simplify the following:
  - Real-time access to flight data.
  - Real-time display of information.
  - Real-time command generation (subject to centralized validation).

  For many application programmers, these functions address the primary functional requirements for flight support (from a user's perspective). For this reason, the Concept Executive will provide additional interfaces which simplify use of these functions.
- Networking - the Concept Executive will provide a complete set of services which allow transparent access to the network(s) of the environment.
- Graphics - the Concept Executive will provide a complete graphics environment. This will include a graphic based user interface, data display mechanisms, and high-level plotting and modeling software.
- Programming languages - the Concept Executive will provide access to system and executive services via a standard programming language. Limited support will be available for additional languages.
- Command line interpreter - the Concept Executive will provide control of the operating system via a command line interface.
- Distributed processing support - the Concept Executive will provide functions which transparently implement the advantages of distributed processing. These include:
  - Fault tolerance and automatic recovery.
  - Automated distribution of processes.
- Events - the Concept Executive will support acquisition, local distribution, and the ability to initiate events.

- System state - the Concept Executive will provide a set of functions which allows the user to query and modify the state of the system.

The described functions will be provided via a set of libraries to be used by programs and commands available for access via the command line interpreter.

5.3.3 Environment Support and Control

The final goal of the Concept Executive is to present an environment which is stable and secure during operations. The target system supports critical flight activity and as such, it is imperative to insure that all system and application software has been fully tested, is the correct release, and corresponds to the current activity. These goals are achieved via:

- Configuration Management - Configuration Management (or CM) guarantees the configuration during software development and during operations. Configuration Management also insures that certain functionality is removed or controlled during operations.
- Security - the security system complements the Configuration Management system, as it is obviously impossible to guarantee the configuration unless users are prevented from purposely or inadvertently modifying it.

Although Environment Support and Control functions may appear to be a burden to application development, these functions provide useful services such as software archive and insuring that up-to-date libraries are used for application development.

5.4 Assumptions

This section discusses assumptions made about the environment and how the Concept Executive will depend upon, interact with, and support systems. In doing so, the scope of the Concept Executive is addressed in more detail.

5.4.1 Assumed Global System Functions

This section addresses functions which are assumed to be performed by external hardware and software subsystems which affect the operation of the entire environment.

5.4.1.1 Simulations

The Concept Executive shall not be responsible for generation of simulation data. The Concept Executive assumes that during a simulation, simulated data is fed into the entire system at a high level which allows all host and workstation-based subsystem functions to be tested in an integrated fashion. In this manner, a simulation proceeds exactly the same as actual operations, except that the data used is simulated.

The Concept Executive also assumes that the simulation system shall have the capability to introduce anomalous data in order to verify that applications can identify such conditions.

5.4.1.2 Network Mode Selection

The Concept Executive assumes that the current mode of the system (development, simulation, operations) will be determined by the state of the global networks. This is logical, as the data and the operation to which it corresponds is what determines the function of the system at any given time.

5.4.2 Data Distribution

This section describes systems responsible for generation and distribution of data to the workstations via the global network(s).

5.4.2.1 Real-time Data Generation

The Concept Executive shall not be responsible for generating the real-time data which is present on the global network(s). The Concept Executive assumes that another system will distribute the data in a broadcast manner such that it is available to all workstations connected to the global network.

The Concept Executive shall assume responsibility for distribution of data to workstations not directly connected to the global network. This may be necessary to support different workstation configurations (such as diskless nodes).

The Concept Executive assumes that another system will provide raw real-time data on this network. The Concept Executive also assumes that if required, another system will generate real-time data which has been decommutated and error checked. This function will be handled by a host or a system separate from the control of the Concept Executive.

5.4.2.2 Real-time Data Retention, Archiving, and Playback

The Concept Executive shall not be responsible for retaining and archiving real-time data for later playback and trend analysis. The Concept Executive assumes that a host will perform this function and will provide functions which facilitate playback of archived data.

5.4.2.3 Generalized Data

The Concept Executive shall not be responsible for generating generalized data which is required by other workstations. The Concept Executive assumes that if various categories of general data are required, they shall be computed on a separate host system and will be made available to the workstations via the global network.

The Concept Executive is responsible for providing interfaces which allow retrieval of general data.

5.4.2.4 Test Activity

The Concept Executive assumes that all forms of data will be available for basic testing purposes during development mode.

5.4.3 Assumed Host Functions

This section describes functions which are the responsibility of a host or cluster of hosts. A host is assumed to provide logically centralized functions or data which is required by all or the majority of workstations.

The Concept Executive makes no assumption as to the architecture or vendor used for the host systems. Such hosts may be based on UNIX or any other operating system. There will most likely be a collection of hosts which run the operating system most appropriate for the associated application.

5.4.3.1 Spacecraft Commands

The Concept Executive assumes that the majority of the functionality relating to spacecraft command generation will be performed on a host system. Validation of spacecraft commands logically resides on a centralized host where commands may be compared to a database of valid commands.

The Concept Executive shall provide an interface which allows real-time transmission of a spacecraft command to the appropriate host. This will be a simple interface which primarily handles the network transmission.

The Concept Executive shall not be responsible for presentation of different interfaces which allow interactive selection of commands. Such interfaces may include manual entry of text or presentation of graphics control panels. Such interfaces will be developed as system applications above the Concept Executive.

5.4.3.2 Network Management

The Concept Executive shall not be responsible for management of the global networks which connect the workstations. The Concept Executive assumes that this function will be performed by a separate host system which analyzes network traffic and identifies problems.

The Concept Executive shall support introduction of interfaces which allow network statistics to be collected for the workstation. This information will be required for network management and for local load analysis.

The Concept Executive shall not be responsible for providing an intuitive interface which allows local users to review the current network status. The Concept Executive will provide interfaces to allow this application to be developed.

5.4.3.3 Health and Status

The Concept Executive shall not be responsible for managing the health and status of the workstations. The Concept Executive assumes that this function will be performed by a separate host or workstation which analyzes statistics from workstations, isolates problems, and interacts with the Concept Executive to take corrective actions. Status assessment is a logically centralized function which relies upon a large database of fault information (expert system) required to diagnose and predict failures.

The Concept Executive shall provide interfaces which allow health and status statistics to be collected for the workstation. This information will be required for global health and status and for local load analysis.

The Concept Executive shall not be responsible for providing an intuitive interface which allows local users to review the current health and status of the workstation. The Concept Executive will provide interfaces to allow this application to be developed.

5.4.3.4 Configuration Management (CM) Workstation

The Concept Executive shall assume presence of an intermediary workstation which is responsible for compiling and loading application programs with certified libraries and software in preparation for certification. The basic process for certification is:

- Develop and test locally an application.
- When ready for final or simulation testing, submit the application source code to the CM Workstation.

- Retrieve loaded executables and perform additional local testing to insure that use of certified libraries did not introduce any problems.
- Upload the application (source and object code) to the CM host for archival and later download purposes.
- During simulation or operations modes, download the required applications.
- Use applications for simulation or operations.

The presence of the CM Workstation significantly affects the requirements of the Concept Executive, as the interaction with the archive host is be handled by the CM Workstation. The Concept Executive is therefore only responsible for communications with the CM Workstation, which is straight-forward as it is a UNIX system.

5.4.3.5 Certified Application Archive

The Concept Executive shall not be responsible for archiving of certified system and application software. A "certified" application is one which has been fully tested and approved for use during operations.

The Concept Executive is responsible for providing interfaces to allow access to the software in this archive.

5.4.4 Executive Assumptions

This section discusses functions provided by subsystems present on the workstation, but which are not within the scope of the Concept Executive. The scope of the Concept Executive is a subjective matter, so justification is provided for why certain functions are to be handled by higher level software.

5.4.4.1 Simplification Functions

The Concept Executive shall not provide a global set of custom simplification functions which provide more convenient interfaces to operating system, networking, graphics, and other functions. The Concept Executive provides an interface which is suitable for use by applications programmers, who are assumed to be satisfied with existing standard interfaces. Past history has proven that attempts to simplify use of UNIX and other interfaces has introduced more problems than were solved.

The Concept Executive simplifies and tailors use of the environment by selecting standard software which provides networking, graphics, and other required functionality. From a functional standpoint, the only custom simplification interfaces provided by the Concept Executive are for data acqusition. This is necessary, as data distribution is critical to the majority of applications developed in the environment.

5.4.4.2 Data Driven Displays

The Concept Executive shall not provide an application which is used to design and manage data-driven displays. Such an application is a requirement in the environment, as textual and graphic presentation of telemetry data is a primary requirement. However, this application is not within the scope of the Concept Executive.

The Concept Executive provides the basic data acquisition and graphics required to support development of a data display application.

5.4.4.3 Custom Command Languages

The Concept Executive shall not provide a custom command language which replaces or masks the native operating system command line interface (the UNIX shell). The Concept Executive specifies use of the UNIX shell for command-based interaction. If a custom language is desired, the Concept Executive presents an interface upon which one could be developed.

5.4.4.4 Local Interfaces to Host Functions

The Concept Executive shall not provide new interfaces to host specific functions. The Concept Executive provides the required network interface to allow data and control to be exchanged between the workstation and the host. The Concept Executive assumes that higher-level applications will provide suitable interfaces for host functions.

5.4.4.5 User Interface

The Concept Executive shall provide for programmatic and command line interfaces to functions. The Concept Executive also provides a basic COTS based user interface upon which applications may be developed. The Concept Executive shall not be responsible for presentation of a graphic user interface to executive functions.

The Concept Executive shall provide interfaces which allow development of graphic user interface applications. The Concept Executive provides all the tools necessary to develop and set the standard for the user interface. The user interface should be one of the first set of applications to be developed as it will establish the basic behavior of all other user interface applications in the environment.

5.4.4.6 On-line Help

The Concept Executive shall not provide a graphic user interface for on-line help. The Concept Executive shall provide on-line help, but this information shall be available via the native command line interface.

6.0 Functional Requirements

This chapter presents the functional requirements of the Concept Executive. These requirements specify a software system which will satisfy the goals of standardization, system integration and application support, and environment control. The functional requirements which satisfy these primary goals are divided into the following:

- Overall System Requirements.
- COTS Subsystem Requirements.
- Custom Subsystem Requirements.

The COTS subsystems specified by the Concept Executive provide a baseline which supports the common development and operational requirements of all applications. The custom subsystems specified by the Concept Executive provide environment control and additional application-specific functionality.

6.1 Overall System Requirements

This section describes functional requirements which express design guidelines, set the direction, or provide functionality which affects the Concept Executive as a whole.

6.1.1 Workstation Based Distributed Processing

The Concept Executive shall provide a distributed development and operations environment which controls and integrates a network of graphic workstations. The Concept Executive shall allow each workstation in the network to function as an application processor for a portion of the ground-based support requirements.

6.1.2 Generic Environment Support

The Concept Executive shall specify requirements at a level which supports multiple ground-based, spacecraft support environments. The target environments include, but are not limited to the following:

- Short duration flights - flights for which the duration is less than 30 days (1 month). This includes support for multiple simultaneous short duration flights. The Concept Executive assumes that in the event of multiple flight support, all spacecraft are very similar.
- Continuous duration flights - long-term flights for which continuous support must be provided.

The Concept Executive shall support environments currently in use and those planned for the future.

6.1.3 Identical Operational Environment

The Concept Executive shall present the same development and operational environment on every workstation in the network. The primary motivation of this requirement is to insure that if one workstation fails, the affected users can utilize any other workstation in the network. By providing a reasonable set of unused workstations, the environment will offer a means of manual fault tolerance.

6.1.4 Workstation Hardware Expandability and Interoperability

The Concept Executive shall offer an extended environment life cycle by insuring expandability and interoperability via the use of software standards. The Concept Executive shall specify a system which is expandable in each of the following ways:

- Addition of new processors - the Concept Executive shall support increased processing requirements via the addition of new workstations. This shall be true to the bandwidth limits of the global communications networks which interconnect the workstations.
- Upgrade of processors - the Concept Executive shall support increased processing requirements by allowing workstations to be upgraded with newer, more powerful components. The Concept Executive shall allow upgrade of processors, disks, graphics devices, other peripherals, and the entire workstation.

The Concept Executive shall allow upgrade of workstations from the same or a different vendor. The Concept Executive shall allow seamless upgrade to any workstation which supports the defined standard software environment.

6.1.5 Real-Time Response

The Concept Executive shall be based on a real-time operating system which has been demonstrated to provide deterministic response adequate to support all time-critical applications in the target environment.

The requirement for real-time response shall be balanced with the subsequent requirements for use of Commercial-Off-The-Shelf (COTS) and standard software. The Concept Executive shall provide real-time response while at the same time utilizing standard software.

6.1.6 Use of Commercial-Off-The-Shelf Software

The Concept Executive shall utilize as much COTS software as is possible within the real-time constraints of the target environment. The Concept Executive specifies use of COTS software in order to:

- Reduce development and support costs.
- Present a standard environment.
- Provide a robust system.

COTS software shall be used to entirely satisfy or form the basis of the majority of the subsystems making up the Concept Executive. The COTS software subsystems specified by the Concept Executive shall include, but not be limited to the following:

- Operating system.
- Executive and application programming languages.
- Command line interface.
- Real-time extensions.
- Networking.
- Graphic user interface (window system).
- Graphics.

The Concept Executive shall not specify any COTS software subsystem which is not a widely available standard and is in turn acknowledged and implemented by a wide variety of workstation vendors.

6.1.7 Use of Standard Software

The Concept Executive, to the extent possible in a real-time environment, shall use software standards. Use of standards will allow the Concept Executive to support a variety of workstations and to easily port to new workstations as they become available. The standard software subsystems shall include, but will not be limited to the following:

- Operating system.
- Executive and application programming languages.
- Command line interface.
- Real-time extensions.
- Networking.
- Graphic user interface (window system).
- Graphics.

All standards specified by the Concept Executive shall be implemented in layers which allow application programmers to select the layer which best suit their convenience and performance requirements.

All implementations of standards used by the Concept Executive shall be verified by a formal test suite which is complete and widely used by the industry. No standard implementation shall be used which does not pass such a test suite.

6.1.8 Migration to POSIX

The Concept Executive shall present a programmatic and interactive interface which is fully compliant with the Portable Operating System Interface Definition (POSIX) standard defined the Institute of Electrical and Electronic Engineers (IEEE). POSIX is a standard which specifies all interfaces to the operating system and related support software (networking, real-time functions, etc.). POSIX shall be used as it is the only interface which is globally accepted and is independent of the direction of any single vendor. This as opposed to offerings from UNIX International (UI) and the Open Software Foundation (OSF) which are driven by certain vendors.

At this time, the entire POSIX standard is not yet defined and it will be several years before the standard is complete. The Concept Executive shall select standards which are compliant with the current direction of POSIX in order to allow a clean migration path in the future. As portions of POSIX are completed, the Concept Executive shall verify that the specified standards remain fully compliant.

6.1.8.1 POSIX Description

The goal of POSIX is to provide application portability across a number of operating systems, including those not based upon UNIX. The complete POSIX standard is defined as follows:

- POSIX 1003.0 - Overall specification of POSIX.

- POSIX 1003.1 - System interface.
- POSIX 1003.2 - Command line interface.
- POSIX 1003.3 - Verification testing.
- POSIX 1003.4 - Real-time interface.
- POSIX 1003.5 - Ada bindings.
- POSIX 1003.6 - Security interface.
- POSIX 1003.7 - System administration interface.
- POSIX 1003.8 - Network interface.
- POSIX 1003.9 - FORTRAN bindings.
- P1201 - X Windows-based graphic user interface (not part of POSIX, but is closely related and is only standard addressing graphic user interfaces).

POSIX is currently not a fully defined standard. At this time, only POSIX 1003.1 is complete. The 1003.1 standard specifies only the programmatic interfaces to the operating system (the equivalent of UNIX system calls). Completed standards are not yet available for command line, network, real-time or other functional interfaces. These interfaces are being addressed by working groups, each of which is at a different level of completion. Although some working groups are nearing completion, it will be several years before the complete standard is available and implemented on a variety of systems.

It is very important to note that POSIX is not an implementation of an operating system. Rather it is a specification of the programmatic and command-level interfaces. This allows for a wide variety of fundamentally different operating systems to provide a POSIX interface and therefore allow development of portable applications.

The reference model for the POSIX interface is the UNIX operating system. The majority of POSIX interfaces closely match existing UNIX interfaces and behavior. Therefore, the most appropriate operating system selection for environments desiring future POSIX compliance is UNIX.

6.1.9 Isolation of All Non-Standard Functions

The Concept Executive shall isolate any and all vendor/hardware-dependent functions, if such functions are required to provide an application support function. If any vendor/hardware-dependent functions are required, the Concept Executive shall prevent a new interface which masks the vendor/hardware dependent functions and therefore will allow development of portable software.

If the vendor/hardware-dependent function represents a function to be present in a forth-coming standard, then the interface presented by the Concept Executive shall mimic the calling sequence and the behavior of the standard function. This requirement may be necessary for support of real-time functions, as there is no standard yet defined.

6.1.10 Executive Portability

The software comprising the Concept Executive shall be very portable. The only portions of the Concept Executive which are not portable shall be those using vendor/hardware-dependent functions in order to provide required application support which is not otherwise available via standard interfaces.

6.1.11 Application Portability

The Concept Executive shall provide an application program interface (API) which allows development of completely portable applications. All system and user applications developed on or above the executive shall be completely portable to other systems which support all standards defined by the Concept Executive. The API will be a collection of standard COTS and custom functions.

The Concept Executive shall prevent (via Configuration Management) application programmers from using (loading) non-standard functions in applications.

6.1.12 Modifications to COTS Software

The Concept Executive shall provide all functionality without requiring modifications to the operating system or any other COTS software. Modification of COTS (MODCOTS) software shall not be required unless there is no other manner in which a required service can be provided. The Concept Executive shall only use modified COTS software after all alternative approaches have been proven ineffective.

Use of modified COTS software is to be avoided as it introduces serious software portability problems. Although one vendor may provide the required modifications, other vendors may refuse or only do so at a significant cost. Use of modified COTS software also introduces additional support costs, as vendors must provide special support for the unique versions of otherwise-COTS software.

6.1.13 Software Subsystem Support

All software subsystems comprising the Concept Executive shall be supported. All COTS subsystems shall be supported by the appropriate vendors. All custom software shall be supported by the developer of the Concept Executive. The Concept Executive shall not use any software which is not fully supported and for which clear responsibility may be defined.

The Concept Executive shall not utilize any unsupported, public domain software. If a public domain, non-vendor supplied software system is absolutely required, then it shall be interpreted as custom software which is the sole responsibility of the Concept Executive developer.

6.1.14 Configuration Independence

The Concept Executive shall be hardware, vendor, and configuration independent. In addition to using standard software, the Concept Executive shall support multiple workstation configurations. The Concept Executive shall not rely upon and base its design on any specific workstation configuration, especially if this configuration is not supported by a variety of workstation vendors. Such a design, even if based on standards, would not allow the goal of interoperability to be achieved.

For more information on configuration independence, refer to the section on distributed processing.

6.1.15 Global and Local Network Independence

The Concept Executive shall be not depend on a specific configuration of the global network(s). The Concept Executive shall function identically whether or not the global network consists of one, two, or more physical networks. The Concept Executive shall mask such details from application programmers and therefore allow the global network configuration to change as needed by new processing requirements or allowed by the introduction of new technology. In particular, the Con-

cept Executive shall support transparent communications over the following global network configurations:

- 1 physical network supplying all real-time and general purpose data.
- 2 or more physical networks which are divided based on the type of data on each network. For example, one network for broadcast, read-only real-time data and another network for multicast and point-to-point, read-write, general purpose data.
- 2 or more physical networks which are divided in order to support multiple concurrent operational activities (multiple simultaneous flights).
- 3 or more networks which are divided to support a combination of the previous two configurations. A reasonable configuration would be one in which one network is used to support all streams of real-time data while 1 additional network is provided for general purpose support of each concurrent activity.

The Concept Executive shall specify network standards to be used for workstation to workstation communications. Ideally, all data output on the global networks will use the same communication protocols. Unfortunately, the Concept Executive cannot define the protocols used by systems generating real-time and host-specific data. In such instances, the Concept Executive shall adapt itself in order to process the different protocols. The Concept Executive shall in turn present all data in a standard manner through a common interface (data acquisition).

6.1.16 Data Throughput Increases

The Concept Executive shall be designed to expand to support throughput processing of increasingly large amounts of data. This is necessary to support the data processing requirements of new on-board systems, multiple simultaneous spacecraft, or even entirely new spacecraft. The Concept Executive shall not use any technology or design which would be obsolete in event of a substantial increase in data throughput requirements. In order to support an extended life cycle, all subsystems comprising the Concept Executive shall allow for order of magnitude increases in the amount of data generated for a spacecraft.

6.1.17 Modes of Operation

The Concept Executive shall not depend on physically separate systems for development and operations. Such a system, while offering some advantages, would be too costly to use due to the duplication of hardware and software. Rather, the Concept Executive shall support different operation states on the same physical networks and workstations. The three different modes of operation are as follows:

- Development - in which application software is developed and preliminary testing is performed. The goal of development mode is to present a flexible and complete environment in which application software may be developed. Development mode will also be used to integrate a new workstation into the network.
- Simulation - in which application software is tested in an integrated fashion with simulated data. The goal of simulation mode is to present an environment which is functionally identical to operations, except that the data (and corresponding flight) is simulated.

- Operations - in which a flight is active and being supported by the system. The goal of operations mode is to present an environment which is well-controlled and in which only certified applications are allowed to execute.

In order to support different operational modes on the same set of physical hardware, different configurations and rules will be followed to define and provide an environment suitable for the current operation. The makeup of each mode will be defined and enforced by the Configuration Management and Security subsystems. The specific requirements for each mode will be discussed in more detail in these sections.

6.1.18 Support of Multiple Modes

The Concept Executive shall support different workstations being in different operating modes. The Concept Executive supports continuous duration and multiple concurrent flights being active on the system (described in the next sections). In each environment, the system has an activity which is in a constant critical operating state. There will never be sufficient down-time (no operational activity) to add new systems or develop and test new software. This creates the requirement for multiple modes to allow new hardware to be introduced and new software to be developed and tested in order to effectively maintain the system.

The Concept Executive shall support a single mode of operation on any given workstation. Support of multiple modes on a single workstation is not reasonable considering security and configuration differences between the modes. That is to say, an individual workstation shall not be able to run different modes simultaneously, but different workstations can be concurrently executing in different modes of operation. Although a given workstation will always be in a single mode, the entire system shall be multi-mode as interpreted by the Concept Executive.

In supporting multiple modes, the Concept Executive shall not allow a workstation in a non-operations mode (development or simulation) to adversely affect the global network or other workstations. The precise capabilities for each non-flight mode are discussed in more detail in the Configuration Management section.

The primary advantage to allowing multiple modes in the system is to support software development during continuous duration or constant, overlapping short-term flight activity. In each instance, it will be necessary to introduce new hardware and system/application software to the environment. This will only be possible if development mode workstations are allowed to effectively develop software, upload the software to central archive, and use simulation mode integrated testing. Multiple modes also allows integration of new workstations and other hardware components into an operational environment. While workstations and new software would not be added to a stable system, this may be required to satisfy new requirements or to replace failed systems.

6.1.19 Concurrent Flight (Operation) Support

The Concept Executive shall provide support for multiple concurrent spacecraft flights at the individual workstation level. Concurrent flights implies that before one flight terminates, another flight begins in such a way that there would not be a time when there are no active flights. Concurrent flight support implies that each individual workstation shall have the capability to simultaneously support multiple flights. In such an environment, The Concept Executive shall not require physically separate workstations to support separate flights.

The ability of the Concept Executive to support concurrent flights in this manner is entirely dependent on the capabilities of the global networks. It is possible that due to network bandwidth or

workstation hardware limitations, that it is impossible to provide a single workstation with all data necessary to support concurrent flights. It is also possible that an entire sub-network of workstations is similarly constrained. Despite this, the Concept Executive shall not impose any workstation-level limitations on the ability to support concurrent flights. The Concept Executive shall support this requirement in the most flexible manner.

The Concept Executive shall allow simultaneous access of data from multiple flights. This allows a user to simultaneously monitor and compare data from similar systems on separate flights. This implies that all data, displays, and commands are tagged with the corresponding flight identifier.

Support of multiple flights on a single workstation will allow users more control over the partitioning of resources dedicated to support of flights. A less flexible solution would be to only allow support of one flight per workstation. This would force the user to log in and out of the workstation to support different flights. An even less flexible solution would be assign sets of workstations to support separate flights (i.e., if there are 10 workstations, 3 could be assigned to one flight and 3 to another). Both solutions are less flexible and would prevent a user from effectively managing workstation resources.

6.1.20 Continuous Duration Flight Support

The Concept Executive shall provide support for continuous duration operations. In order to provide this type of support the Concept Executive must allow for the integration of new workstations and development of new system and application software. These requirements are supported by the multi-mode capability of the Concept Executive.

6.1.21 Programmatic and Command Level Interfaces

The Concept Executive shall present a complete programmatic and command level interface. The Concept Executive shall provide a set of ANSI C language compatible libraries to be used by application programmers. The Concept Executive shall provide a set of commands which are compatible with the UNIX command line interpreter.

The Concept Executive also provides a completely COTS based graphic user interface which forms the basis for the interface presented on the bitmapped workstation displays. The Concept Executive shall provide an interface whereby the UNIX command line interpreter is executed in this environment and therefore allows execution of commands provided by the Concept Executive.

6.1.22 Support of ASCII Terminals

The Concept Executive shall support limited use of ASCII terminals. ASCII terminals are an inexpensive means of interfacing with the system to perform system maintenance and support of the workstation graphic displays. Concept Executive provides commands which are compatible with the native UNIX command line interpreter. This supports limited use of ASCII terminals. "Limited" support of ASCII terminals shall be interpreted as all interfaces and commands which are not associated with display of data.

6.1.23 Resource Utilization

The custom subsystems of the Concept Executive shall make efficient use of all standard functions available through the operating system and other COTS software. The Concept Executive shall not require unreasonable system resources.

During a mode of operation in which the Concept Executive is initialized, a user is currently logged in, but no activity is taking place, the Concept Executive shall not use more than an average of 5 percent (5%) of the total available processing, network, or peripheral access capability. In measuring this value, the period in which the average is taken shall be at least 15 minutes to insure that the measurement is not skewed.

6.1.24 Application Termination

During normal or abnormal termination of a Concept Executive custom application, all necessary clean up actions shall be performed. This includes, but is not limited to deallocation of dynamic resources, proper closing of files, and clean up of all external data structures semaphores, shared memory, etc.).

After normal or abnormal termination, it shall be possible to restart the failed process. Failed processes shall not terminate in such a manner which prevents subsequent use.

6.1.25 Error Handling and Reporting

In event of an error, the Concept Executive shall take the appropriate action. A non-fatal error shall not cause any Concept Executive process to terminate.

All serious errors shall be logged by the Concept Executive. A log file shall be maintained for analysis of such errors.

6.1.26 On-line Help

The Concept Executive shall provide on-line help for all COTS and custom interfaces. This includes all programmatic and command line interfaces. The on-line help shall be compatible with the native UNIX help command ("man"). All help text shall conform to the format established for UNIX "man" pages.

6.2 COTS Subsystem Requirements

This section discusses requirements which are satisfied by COTS software products. All COTS subsystems represent industry standards which will be available on a wide variety of workstations. The COTS subsystems combine to specify the basic operating environment upon which the custom Concept Executive subsystems will be developed. The COTS subsystems within the Concept Executive are as follows:

- Operating system.
- Programming languages.
- Command line interface.
- Networking.
- Graphic user interface (window system).
- Graphics plotting and modeling.
- Real-time extensions.

Each COTS subsystem will be described in more detail in the following sections. For each subsystem, the Concept Executive will specify all requirements and then will recommend a specific standard.

6.2.1 Operating System

The Concept Executive shall be developed in and provide access to the UNIX operating system. Use of UNIX is assumed, as it is the primary operating system used on engineering-class workstations. UNIX is widely viewed as a standard operating system, provides a very powerful development environment, and provides a clean migration path to POSIX (as POSIX is based on a UNIX interface). The only question concerning use of UNIX is which variant and release shall be used.

6.2.1.1 Recommended Standard

At this time, it appears that two variants of the UNIX operating system will compete for market share. These are System V Release 4 (SVR4) from UNIX International (UI) and OSF/1 from the Open Software Foundation (OSF). SVR4 is a merge of the best (in the interpretation of UI) features of System V Release 3, Berkeley Software Distribution (BSD) 4.3, and Xenix. SVR4 also includes functionality from Sun's SunOS operating system. OSF/1 is primarily based on the Advanced Interactive Executive (AIX) operating system developed by IBM. OSF/1 will also use features from Mach for multiprocessor support. Mach is an operating system designed at Carnegie Mellon University.

Development of these two operating systems is welcome in that it provides a means of merging the functionality of System V and BSD variants of UNIX. It is equally unfortunate that two organizations are simply recreating the problem by developing two new operating systems. There is no clear decision in selecting one of these operating systems as the standard for the Concept Executive. Both operating systems provide surprisingly similar functionality, conform to POSIX, and include mechanisms for real-time.

Despite the confusion, the Concept Executive shall be developed in and provide access to the System V Release 4 (SVR4) version of UNIX. SVR4 was selected in preference over other UNIX variants for the following reasons:

- SVR4 will be available in early 1990, as opposed to the OSF/1 operating system offering from OSF which will not be available for some time.
- SVR4 offers a combination of the best features of the three operating system variants. SVR4 also includes a number of features taken from Sun's operating system (SunOS).
- Applications developed in System V, BSD, and Xenix will be fully compatible with SVR4.
- SVR4 satisfies the upgrade path to POSIX.
- SVR4 will offer real-time capability.
- SVR4 is based on robust features, where as OSF/1 is based on unproven technologies.

One disadvantage of SVR4 is that it will include the OpenLook user interface. While this is a useful package, it is not specified by the Concept Executive. The Concept Executive will use Motif, as is explained in a later section. Although Motif is from OSF, it is more widely accepted and viewed as a superior product. Motif is also available separately from the OSF/1 operating system and will appear on many systems supporting SVR4.

SVR4 provides a great deal of functionality. In summary, some of the significant features in SVR4 include the following:

- Standards Support:
  - POSIX 1003.1 compatibility.

- ANSI C.
- TCP/IP, telnet, ftp, and smtp.

- System V Release 3.2 features:
  - Interprocess communications facilities (shared memory, message queues, semaphores).
  - Streams.
  - uucp.
  - Transport Level Interface (TLI).
  - Remote File System (RFS).
- BSD features:
  - C shell.
  - Fast file system.
  - Selected commands and system calls (including signals).
  - Symbolic links.
  - Sockets, remote commands (r* commands) and, inetd.
- SunOS features:
  - Memory mapping interface from SunOS.
  - Network File System (NFS).
  - Remote Procedure Calls (RPC).
  - EXternal Data Representation (XDR).
- Xenix compatibility.
- New features:
  - Streams enhancements.
  - Internationalization.
  - Virtual File System.
  - Korn shell.
  - Process file system.
  - Message management.
  - Dynamic linking (shared libraries).
  - Network selection.
  - Name to address mapping.
  - Mail.

Selection of SVR4 is a logical extension of System V Release 3 which was specified in the HISDE prototype. Applications developed in System V Release 3 should port easily to SVR4.

6.2.2 Programming Language Interface

The Concept Executive shall be developed in and support development of applications in a widely available and standard programming language. The programming language used and supported by the Concept Executive shall:

- Be established as an industry standard and support development of portable applications.

- Support efficient development of a wide variety of system and application programs.
- Function well in a UNIX environment and support interfaces (bindings) to all COTS and custom Concept Executive functions.

The programming language specified for the Concept Executive is described in the following section.

6.2.2.1 Recommended Standard

The Concept Executive shall use and provide access to the ANSI C language. ANSI C is becoming widely available and should be adopted as a standard in the very near future. ANSI C has been selected as the standard due its portability and industry acceptance. Applications developed in ANSI C are normally easier to port than programs written in older Kernigan and Ritchie (K&R) C. In summary, ANSI C offers the following advantages:

- Compatibility with K&R C - new ANSI C compilers will support applications developed in K&R C.
- Standard libraries - ANSI C includes a specification of libraries which implement basic language extensions (input/output, string manipulation, etc.).
- Additional portability - ANSI C specifies a set of headers (include files). This is a major improvement for portability of applications, as K&R did not specify headers and in many instances, header files varied between vendors. The ANSI committee resolved these differences into one standard that is not controlled by vendors.

ANSI C had not been formally accepted as a standard at the time this document was generated. All technical details have been resolved, but procedural delays have prevented the standard from being formally released. Nevertheless, industry has widely accepted ANSI C as the C standard. Acceptance is demonstrated by the delivery of compilers based on the ANSI C draft standard and the adoption of ANSI C in SVR4.

6.2.2.2 C Versus Ada

Specification of the ANSI C language rather than Ada is a difficult decision. This selection is rationalized by the following advantages of ANSI C.

- POSIX compatibility - the interface model for POSIX is ANSI C.
- Performance - C provides a finer level of control than possible with Ada. Proper use of C will allow smaller, more efficient programs to be developed.
- Availability of standards - one advantage to using Ada is that the interfaces for features such as tasks (threads) and I/O are standardized. In the future, such features will be standardized by POSIX, thus eliminating this advantage.
- Interface bindings - in a UNIX environment, most software systems (graphics, networking, etc.) only provide C language bindings (function calls). It is not possible to write an ADA program which directly calls such functions.

In the future, POSIX will define an environment which supports integrated use of Ada and other languages. This however will occur several years in the future.

6.2.2.3 Additional Language Support

The Concept Executive shall select standards and COTS software which supports application development in each user's preferred language. Ideally, the Concept Executive shall provide bindings to all system functions for several standardized languages. However, this is not a feasible requirement at the current time. As described above, the Concept Executive shall only provide access to all system interfaces via the ANSI C language.

Although the Concept Executive does not provide a complete set of interfaces, it does not preclude an application programmer from developing software in other standard languages (such as Ada and Fortran). Such languages may be freely used in the environment as long as the applications do not depend on interfaces which may not be present on other systems. Use of other programming languages shall be limited to isolated functions or stand-alone programs which do not require or use system interfaces. For example, a C program could be developed to display graphics for calculation results generated in Fortran functions.

In the future, the Concept Executive shall provide complete access to all system functions for the following languages:

- Ada.
- FORTRAN.

It is expected that the POSIX 1003.5 and 1003.9 working groups will drive the movement for support of these languages within a POSIX environment.

6.2.3 Command Line Interface

The Concept Executive shall provide a command line interface to allow interaction with the operating system, execution of commands, and generation of command line programs (shell scripts). To satisfy these requirements, the command line interface used could be any one of the following:

- The native operating system command line interpreter (the UNIX shell).
- A new command line interpreter which uses the current command paradigm.
- A completely custom language and command interface (either line or graphics based).

Providing either type of custom language interface has advantages in that it can present commands which are tailored to a specific application environment, are suited for the targeted users of the system, and have embedded control mechanisms for the environment. Development of a complete, high-level language is a reasonable goal, but one which will require a significant amount of effort. The disadvantages of providing a custom language are as follows:

- The new language would be unfamiliar to users.
- The operating system interface (the UNIX shell) is known and preferred.
- Creating a new custom language would entail a significant amount of time and effort to develop.
- Once developed it would add overhead to the command process and slow the response time.

The Concept Executive shall provide the user with a command-level interface by allowing access to the native UNIX command line interpreter (the shell). The Concept Executive shall depend on this COTS command line interface provided by the selected operating system (SVR4). The Con-

cept Executive shall not attempt to redefine this interface and present a new command level language.

The primary disadvantage of the UNIX shell is that its use is difficult to control. Once the user has access to the command line, it is possible to execute any command and access any data available on the system (subject to permissions). This problem will be addressed by Configuration Management, which will prevent misuse of commands and data by defining the system file and permissions configuration.

The Concept Executive does not discourage nor preclude development of a higher-level software subsystem which provides a new command interface. The Concept Executive provides the basic functions upon which this new interface could be developed. The new interface could address the command requirements of the entire environment or be tailored to a specific application (such as display management or computation generation).

The combination of the command line and the Concept Executive user interface (explained in a subsequent section) provide an interface which can be expanded to suit the requirements of the environment. As is, the Concept Executive provides a window-based user interface for which the basic command interface is the UNIX shell. By developing higher-level software, an environment can introduce a custom command language interface and/or an intuitive window/icon based interface.

6.2.3.1 Recommended Standard

The operating system specified by the Concept Executive (SVR4) shall provide access to the following command line interpreter (shell):

- Korn shell - a relatively new shell which provides a superset of the functionality found in the Bourne shell. The Korn shell offers a number of new functions and is completely compatible with the Bourne shell. Note that by providing the Korn shell, SVR4 indirectly provides the Bourne shell.

The Korn shell is compatible with the Bourne shell, is more efficient than both the Bourne and C shells, and provides interactive features similar to those found in the C shell. In short, the Korn shell offers a command environment with all the advantages of the Bourne and C shells. Use of the Korn shell is also consistent with the shell specification in POSIX 1003.2.

6.2.4 Networking

The Concept Executive shall provide support for the networks connecting the workstations. The Concept Executive shall provide network communications interfaces which isolate workstation and global network configuration details and allow development of portable applications. This support shall not depend on the workstation configuration. The Concept Executive shall include support for the following network services.

- Connection based communications - a workstation shall be able to transfer data to all workstations on the network.
- Distributed processing - a workstation shall be able to distribute processing loads to other workstations within the network.
- Network/workstation status - status of a workstation and network interfaces shall be available to other workstations in the system.

- Electronic mail - a workstation shall be able to generate, deliver, read, print, and delete mail messages. All messages shall be time stamped and allow message replies. A visual indication shall be provided to users upon receipt of mail and optionally upon reading of messages sent.
- Directory services - workstations shall provide logical names to physical resources. This data shall be available to all workstations on the network. All mappings between logical names and physical resources shall be provided in an ASCII table.
- Remote login - remote login shall be available on a limited basis to support system administration. This function shall be compatible with Configuration Management.
- Remote file sharing - remote file sharing shall be available so a complete environment need not exist at all workstations. The file sharing available to a workstation shall be controlled by Configuration Management.

All network communication software shall adhere to the Open Systems Interconnect (OSI) seven layer model. Each requirement will be addressed by specifying a standard which represents 1 or more layers in this model.

6.2.4.1 Recommended Standards

The Concept Executive network communications software shall be POSIX 1003.8 compliant. This document is still in draft form, but the various layers are based on ISO/OSI or existing IEEE standards. The layered approach of ISO 7498 (the classic OSI model) is used. Figure 6-1 summarizes the OSI layers as referenced in POSIX 1003.1.

Layer	Description	Standard	Title
7	Applications Layer ASCE FTAM Mail/Message Job Transfer	 ISO DP 8649 ISO DP 8650 ISO DP 8571 CCITT X.400 ISO DP 8831 ISO DP 8832	 Definition of ASCE Specification of ASCE Specification for FTAM Message Handling System Definition of Job Transfer Spec. for Job Transfer Spec.
6	Presentation Layer	ISO DP 8822 ISO DP 8823	Connection-Oriented Presentation Service Def. Connection-Oriented Presentation Protocol Spec.
5	Session Layer	ISO 8326 ISO 8327	Basic Connection-Oriented Session Service Def. Basic Connection-Oriented Session Protocol Spec.
4	Transport Layer	ISO 8072 ISO 8073	Transport Service Def. Connection Oriented Transport Protocol Spec.
3	Network Layer	ISO 8348 ISO 8473	Network Service Definition Protocol for Providing Connectionless-Mode Network Service
2	Link Layer Control	IEEE 802.2	Logical Link Control
1	Physical Layer	IEEE 802.3 IEEE 802.4 IEEE 802.5	Ethernet Token Bus Token Ring

Figure 6-1 Network Software Hierarchy

POSIX has been chosen as the networking standard for portability between vendors. It conforms to the OSI model, so layers are independent with standard interfaces between layers. This keeps from being locked into one technology or vendor. This also keeps more options open when new technologies become available, such as Fiber Distributed Data Interface (FDDI) at layer one.

POSIX uses ISO standards as the bases for all layers. This will result in a wider range of industry support. At the current time the industry is still developing ISO compliant products, but the expected demand for ISO compliant layers will spur further development.

Layer 7 protocols shall be used to fulfill the requirements for the Concept Executive. If layer 7 protocols do not provide the necessary functionality, layer 4 shall be used. Below is a summary of the layer 7 protocols:

- File Transfer and Management (FTAM) - provides services for a virtual file system. This functionality includes the creation, manipulation, and deletion of files. Also, single records within files may be accessed using FTAM.
- Job Transfer and Manipulation (JTM) - is designed to support computer to computer communications for the purpose of performing work remotely.

- CCITT X.400 - provides a standard for mail formatting and routing. Several COTS packages exist that conform to X.400.
- Virtual Terminal Protocol and Service - provides a standard way to map terminals to actual data links and a standard means to establish virtual terminal connections. COTS packages based on this standard provide remote login capability.
- CCITT X.500 - provides directory services to map logical names to physical devices.
- ACSE (formerly CASE) - allows the communication channels to be opened, maintained, and released. These services are used by other layer seven applications such as FTAM.

At the time of this report, ISO does not support remote file sharing. POSIX is considering a standard remote file sharing standard, but this will not be available in the near future. Since SVID is an interim operating system, the Concept Executive shall use network file sharing (NFS) to share files between workstations. This is consistent with the Government Open Systems Interconnection Profile (GOSIP) for remote file sharing.

6.2.5 Graphic User Interface

The Concept Executive shall specify a standard graphic user interface (GUI) system which is the basis for all user interaction on bitmapped workstation displays. The graphic user interface shall consist of a number of standard COTS libraries and applications which present an environment suitable for development and use of custom applications.

The graphic user interface provided by the Concept Executive shall be based upon the following:

- Standards - the entire graphic user interface shall be based on a well-defined standard. All interfaces at all programmatic and interactive levels shall be standardized.
- Client-server model - the graphic user interface shall be based on the client-server model, in which the server is a process which accepts standard graphics requests and implements them in the most efficient manner with the available hardware. The server manges and controls interaction for any number of clients. The separation of server and client allows the two to reside on separate physical systems.
- Network transparent - clients shall be able to communicate with the server over the communications network. The only transport requirements are that the mechanism be reliable (guaranteed delivery, proper order, and no duplication). The transport mechanism may be a fast medium (shared memory) for local access or any reliable transport protocol for network communications (TCP/IP or OSI).

The Concept Executive graphic user interface shall also provide a layered collection of interfaces which allow efficient development of graphic user interfaces. Each interface layer shall be based on and compatible with all lower layers. Use of several layers allows applications to interface in a manner consistent with performance and development requirements. The interface layers provided shall include the following:

- Low-level interface - a direct programmatic interface to the low-level graphics functions which are the basis for the graphic user interface.
- User interface toolkit - a higher-level programmatic interface which implements basic user interface objects such as, but is not limited to menus, labels, text fields, buttons,

scrollbars, and popup-windows. This toolkit shall be expandable to allow application programmers to add new, application-specific objects.

- User interface language (UIL) - a separate high-level language which allows development of graphic user interfaces. Use of the UIL allows graphic user interfaces to be developed and modified without updates to actual C code.
- Look and feel specification - a basic specification of how a user interface application interacts with the user. This specification standardizes menu appearance, use of keyboard, colors, and other aspects of interaction. Use of a standard look and feel minimizes training costs and reduces interaction errors.

The Concept Executive graphic user interface shall also provide a set of applications which form the basis for the interactive environment. This set of applications shall include:

- A window manager - this application allows the user to manipulate any and all windows on the workstation display. The window manager provides functions such as, but not limited to resize, move, iconify, raise, hide, and kill.
- Interactive applications - in order to function in a graphic user interface, the system must provide a set of applications which provide basic interaction services. This set of applications works with the window manager to provide the basic "desktop" environment. This set of applications includes, but is not limited to terminal emulation, bitmap drawing, font creation and maintenance, environment control, display customization, window information, and display dumps.

The following section discusses the standards which have been selected for these requirements.

6.2.5.1 Recommended Standards

The foundation for all graphics used and provided by the Concept Executive shall be the X Windows standard defined by X Consortium at the Massachusetts Institute of Technology (MIT). The current X Windows system is Version 11 Release 4. This standard, as defined by the X Consortium includes the following specifications and libraries:

- X protocol specification - the protocol which is the core of the X Windows system. The X protocol is implemented in a server and accessed via a layered set of C language libraries.
- Xlib library - a C language interface which provides a low-level implementation of the X protocol.
- Xt library - a C language interface which provides the mechanism for developing graphic user interface objects (widgets).

The X Windows system satisfies the requirements of standardization, client-server model, and is network transparency. The Xlib and Xt library satisfy the requirements of a low-level library and the foundation for a toolkit. This however does not represent a complete environment, so the Concept Executive shall specify additional standards.

At this time, there is no industry wide standard for higher level programmatic and interactive interfaces which are based on X Windows. Two directly competing organizations (UNIX International (UI) and Open Software Foundation (OSF)) are proposing different standards. UI is proposing the OpenLook interface while OSF is proposing Motif. Each interface satisfies several of the Concept Executive requirements and has inherent advantages and disadvantages. It is again unfor-

tunate however that the industry cannot decide on a single interface. Fortunately, the IEEE P1201 committee is attempting to establish a single standard. The P1201 standard will select one or a combination of both. Until this time, the Concept Executive shall select the "standard" which is most likely to become widely accepted.

The Concept Executive shall specify use of Motif to provide several higher level interfaces. This is a difficult decision, as although OpenLook is to be part of SVR4, Motif is viewed as a better interface. Motif will satisfy several remaining requirements by providing the following:

- Widget set - a complete, robust set of widgets with an attractive 3-dimensional look. This widget set is widely viewed as the best in the industry.
- User Interface Language (UIL) - a language which allows specification of user interfaces. A UIL compiler application is included to generate a data file from a UIL program. This data file is used inside an actual program to generate the user interface.
- Window manager - a powerful and flexible window manager which is consistent with the appearance of the widget set.
- Look and feel specification - a specification of a look and feel behavior which is acceptable and is compatible with IBM's presentation manager environment used on personal computers.

The following figure illustrates the relationship of the various user standards specified by the Concept Executive. This figure includes the GKS and PHIGS standards described in the next section.

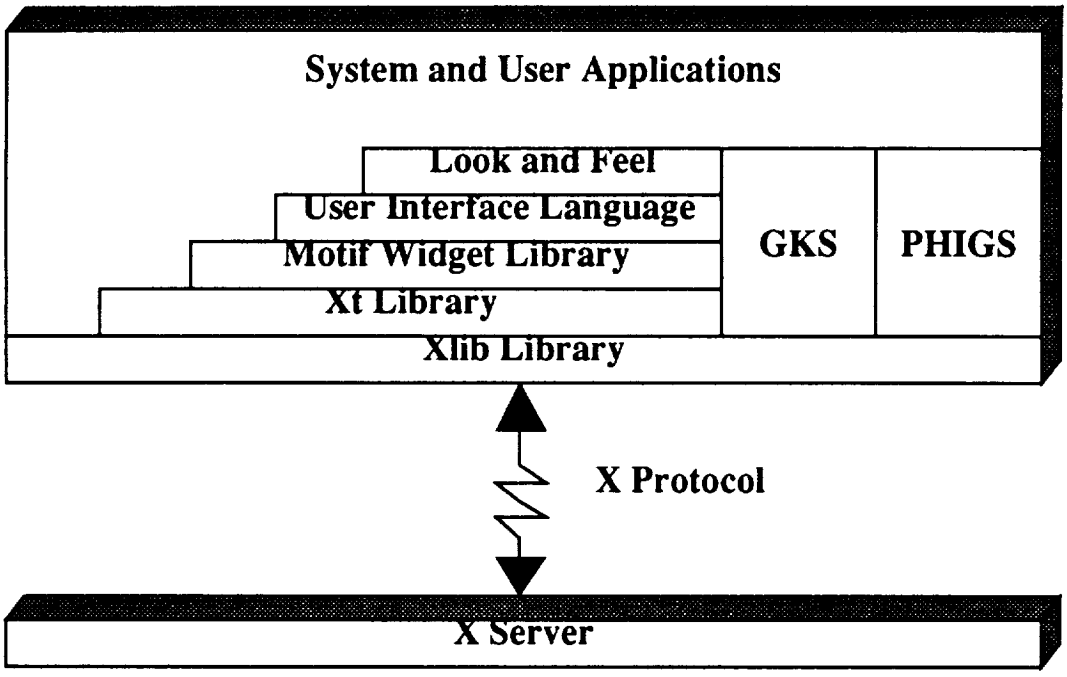


Figure 6-2 Relationship of Graphic User Interface Standards

The only remaining requirement which is not addressed is that of interactive applications. Neither the X Consortium nor Motif specifies a suite of interactive applications which may be used as the basis for normal user interaction. Most X Window system implementations provide a small set of applications which are modeled after those provided by the X Consortium. However, this set of

applications is not a standard and there is no guarantee that they will be provided with any given workstation system.

At this time, there is no way to specify a standard for this requirement. Therefore the Concept Executive shall specify that the interface be composed of the basic applications provided by the X Consortium and provided with most X Windows implementations. This set of applications includes:

- xterm - terminal emulator used to allow normal access to UNIX shell.
- xset/xsetroot - used to set individual and root window characteristics.
- xhost - used to disable and enable remote clients to access server.
- xwd/xwud/xpr - used to dump, display, and print window images.
- xlsfonts/xfd - used to list available fonts and to display a font.
- xdpyinfo - used to obtain information about the current display.
- xkill - used to kill a window and the attached process.
- xwininfo/xlswins - list windows and window information.
- xmodmap - used to modify the keyboard mappings.
- xrdb - used to store resources in the X server.
- xrefresh - used to refresh (clean up) the display.
- bitmap - used to develop bitmaps used for icons and images.

There are additional clients such as clocks, calculators, etc., which are useful, but not required in the environment.

6.2.6 Graphic Plotting and Modeling

The Concept Executive shall support integrated use of specialized graphic plotting and modeling software. The basic function of the graphic user interface is to present and manage user interfaces. While it supports basic drawing operations such as lines, circles, and boxes, it does not support complex 2 or 3-dimensional rendering, in which multiple coordinate systems, transformations, and other functions are required. Such software is required to support development of complex plots, graphs, models, and transformed images. The Concept Executive shall specify a software system which provides these functions and is:

- Integrated with the graphics user interface - the specialized graphic software shall be compatible and will behave properly within the graphic user interface system. The graphics software will be usable with graphic user interface functions within the same application. This allows the graphic user interface software to be used to generate the user interface and the specialized graphics software to generate the desired plots or models.
- All graphics shall be rendered through the same mechanism as used for the graphics user interface. This insures that advantages of portability and network transparency.

6.2.6.1 Recommended Standards

The Concept Executive shall provide graphic plotting and modeling functionality via the Graphics Kernel System (GKS) and/or the Programmer's Hierarchal Interactive Graphics System (PHIGS) standards defined by the International Standards Organization (ISO). Both graphics systems are widely accepted and available on a number of workstation systems.

GKS is intended to support static, 2-dimensional graphics. GKS is relatively fast (as compared to PHIGS) and is available in a number of implementations which are compatible (as described above) with X Windows.

PHIGS (and PHIGS+) is intended to support dynamic, 3-dimensional graphics. PHIGS is most commonly used in environments requiring very high-performance, 3-dimensional rendering. Most PHIGS implementations also offer 2-dimensional functionality. The major disadvantage of PHIGS is that due to 3-dimensional and dynamic update capabilities, it is slower than GKS. Most PHIGS systems currently available are also not compatible with X Windows.

PHIGS is a more functional system than GKS and consequently, involves more overhead even for simple operations. For an environment which is strictly 2-dimensional, GKS will offer better performance and adequate functionality. For environments with more demanding graphics requirements, PHIGS is a better choice. Another advantage to PHIGS is that the X Consortium is planning a standard extension called PHIGS Extensions to X (PEX). PEX will provide full PHIGS+ functionality directly in the X protocol.

Although desirable, it would be very expensive to specify both GKS and PHIGS for the Concept Executive, as each would have to be present on all workstations. This is true even if only a few users required one of the graphics systems, as both systems would be required on each workstation to provide a consistent environment. A better solution is to select one of the systems for each environment to which the Concept Executive is applied.

6.2.7 Real-time Extensions

The Concept Executive shall be developed in and provide a complete set of real-time functions. The implementation of the SVR4 operating system must provide real-time interfaces which can be demonstrated to provide the required responses.

The real-time functions provided by the Concept Executive shall include, but not be limited to the following:

- Priority scheduling - allows assignment of real-time priorities which insure that critical processes execute when necessary. This is opposed to the normal UNIX scheduling algorithm in which CPU-intensive processes have their priority "aged" to improve the response of other processes.
- Asynchronous event notification - allows a process to keep track of multiple asynchronous events that are performed in parallel. An asynchronous event may be generated as a result of a timer, message arrival, or any user defined event.
- Message passing - an interprocess communications facility that allows passing and queueing of messages. A real-time mechanism for passing only the address of data shall be available.
- Process memory locking - allows a process to be locked into core (as opposed to virtual) memory to prevent swapping.

- Shared memory - allows processes to share data through common memory.
- Binary semaphores - a high-performance process synchronization mechanism.
- Threads - a simple and efficient mechanism for establishing a separate flow of control. Threads are more efficient than processes and are useful for handling asynchronous functions.
- Timers - allow a process to set up periodic, offset, relative, and absolute timers.
- Real-time synchronous and asynchronous I/O - allows for real-time I/O for unbuffered or buffered data.
- Real-time files - allows an application to optimize file access times by indicating the type of file accesses at file creation time.

6.2.7.1 Recommended Standard

At the current time, there is no standard for real-time interfaces. The few UNIX operating systems which provide real-time extensions do so in a manner which is proprietary and vendor-dependent. This is a serious problem as real-time features are required throughout the Concept Executive and will be required by system and application programs.

The only proposed real-time standard is POSIX 1003.4, which specifies real-time extensions. This standard is currently in balloting, but even if the standard is accepted, it will be some time before implementations are available. It is expected that SVR4 will provide a POSIX 1003.4 interface for existing real-time functions. It will take longer to provide all the real-time functionality that POSIX 1003.4 specifies.

In the interim, the Concept Executive shall utilize the real-time features of the selected operating system and provide POSIX-like interfaces which mimic the behavior of the POSIX 1003.4 functions.

6.3 Custom Software Requirements

This section discusses requirements which are satisfied by custom software subsystems. These custom subsystems will be developed using the COTS features provided by the Concept Executive. The Concept Executive provides a relatively small set of efficient custom interfaces. These interfaces integrate the environment and support development of applications which must access flight-generated data. The custom subsystems provided by the Concept Executive include the following:

- Configuration Management.
- Security.
- System State.
- Data Acquisition.
- Events.
- General-Purpose Communications.
- Distributed Processing.
- User Interface.

Each Concept Executive custom subsystem will be described in more detail in the following sections.

6.3.1 Configuration Management Subsystem

The Concept Executive shall provide a Configuration Management (CM) subsystem which will guarantee the configuration of a workstation during operations mode. This subsystem shall provide:

- Commands to handle the submission, retrieval, upload, download, etc. of all certified software.
- Access to certified libraries which are guaranteed to be safe for operations.
- A completely certified environment for operations mode.
- A method for downloading a list of certified files.
- Protection for the global networks during development and simulation modes.
- Restricted command access during operations mode.
- Protection of the Concept Executive during execution.
- Controlled access to the root permission.
- Method for integrating new workstations and software revisions into the network.

The CM subsystem shall verify that only certified software is downloaded to a workstation for operation. To develop a certified application, the user shall interact with the CM workstation and CM host. The following functions will be provided for this purpose:

- Submit an application job to the CM workstation for compilation and loading with certified libraries
- Return a copy of the generated executables from the CM workstation.
- Obtain status of application job submitted to CM workstation.
- List the files for an application job which is active on the CM workstation.
- Cancel an application job active on the CM workstation.
- Upload an application from the CM workstation to the CM host.
- List the files that are currently under configuration management on the CM host.
- Specify a file or list of files to be downloaded from the CM host
- Specify a certified library to be downloaded from the CM host.

Figure 6-3 illustrates the path of an application from local user workstation, to CM workstation, to the CM host, and back to the local user workstation.

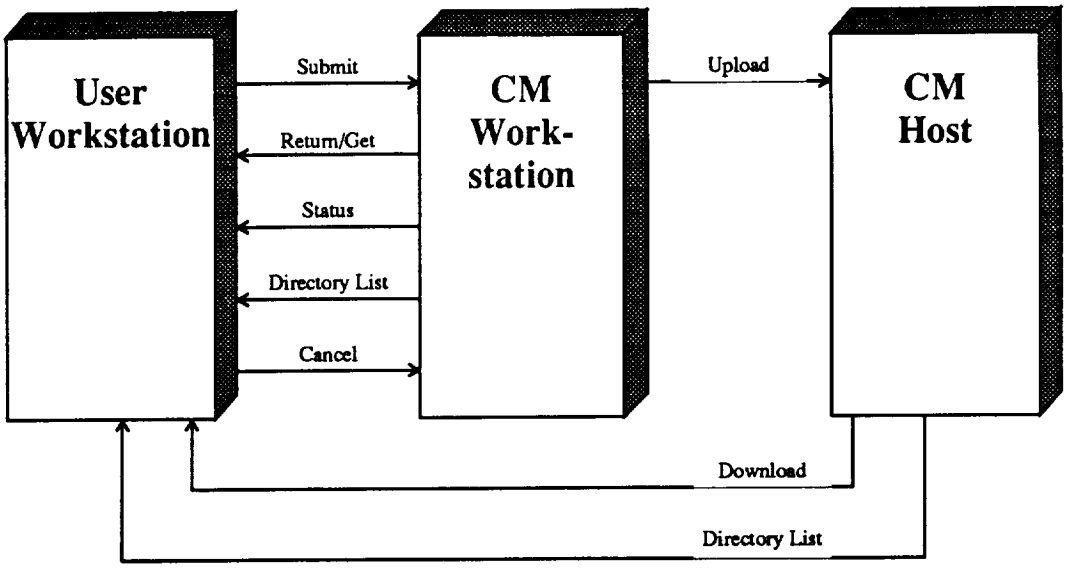


Figure 6-3 Configuration Management Data Flow

The CM subsystem shall provide access to certified libraries which have been insured to be free of viruses, errors, and are up-to-date. Once a source file has been tested and debugged it will be submitted to the CM workstation. The CM subsystem shall require that a makefile be submitted with the application source files. The CM workstation will then be responsible for compiling the source file with the certified libraries using this makefile. Prior to the compilation on the CM workstation, the integrity of the source code content shall be verified by performing a limited automatic search for unusual coding practices. Once compiled it shall be returned to the workstation for testing to ensure that the certified libraries did not induce any problems into the execution of the application. All applications will have to be loaded with these libraries before they can be certified. Once an application has been certified it shall be uploaded to the CM host.

When a user has logged into a session on a workstation, the mode of operation shall be checked by the CM subsystem. If the mode of operation has changed, the CM subsystem shall clear all operating system and executive directories prior to download and all executive and operating system software shall be recycled for the selected mode. If the selected mode is "Operations", then all operational directories will be cleared prior to download and only those directories will be accessible during actual operation. Once the directories have been cleared, the user's operational software shall be downloaded from the CM host, thereby guaranteeing that only up-to-date, certified software shall be executed.

The CM subsystem shall allow the download of user-specific applications by identifying the applications individually or in a file which lists all desired applications. During login the CM subsystem will check for this file and use it to automate the download procedure, if it exists. Use of this download list file will help to decrease the amount of time necessary to begin a session in operations mode.

Updates to the operating system and the executive software shall be downloaded to all workstations simultaneously during scheduled down time for system maintenance. This will insure that all workstations at all disciplines are operating under the same system software revisions. After a revision of software has been installed, a simulation exercising all certified software which is depen-

dent on the updated software (operating system or executive) must be run to insure that there are no backward dependencies to previous revisions of software. All applications which would possibly be affected by a software update should be maintained by the CM host. These applications should be flagged for re-certification whenever a software update occurs.

If a workstation is in development mode, the CM subsystem shall not allow any writing to be done from the workstation onto the global networks. This will prevent any unverified commands from being exported to an actual flight. During simulation mode, the global networks will be accessible for writing, but the CM subsystem will provide a mechanism which will prevent modification to an actual flight which may be executing. This mechanism may be a special flight number which is reserved for simulations, such as STS-00. During development mode, the CM subsystem shall allow for the modification and compilation of programs. However, during operations mode, access to text editors and compilers shall not be allowed. During simulation and operations modes, the CM subsystem shall insure that only certified software is executed. During development mode the CM subsystem will not restrict the window environment of the user. During simulation and operations mode, however, the user's window environment will be governed by the xterm, window manager, and shell supported by the Concept Executive.

All files relevant to the control of a mission shall be managed by the CM subsystem. This includes load files, control tables, and data files, as well as source files. No data files shall be allowed to be read from a secondary device onto a workstation in operations mode.

Depending on the selected mode of operation, the CM subsystem shall provide access to the appropriate versions and configuration of software and data. During operations mode these commands and functions shall be restricted via tailored file systems. For example, a user in operations will not be allowed to edit, compile, or load a source file.

Configuration management shall be provided through the use of file permissions, limited use of the root permissions, and the aforementioned tailored file systems for different modes of operations. Access to the operating system shall be provided through the standard operating system command line interface, but shall be restricted by specially loaded file system containing only those commands allowed for a particular mode of operation.

There should be no way for a user to exit from the Concept Executive on a workstation unless they have access to the root password. As a security issue, access to root shall be reserved only for system administration users.

A new workstation shall not be connected to the network without having the current revision of the Concept Executive loaded. This is a procedure which cannot be enforced by the CM subsystem but is nonetheless important to the configuration management of the system.

During the support of concurrent multiple flights, if different flights require different revisions of software, there needs to be a mechanism for executing only software certified for the concurrent flights. This could be handled by scheduling down-time for system maintenance. At specified intervals, there should be a period of time where there are no active flights. During this period of time all operating system and executive software could be updated. This would force all concurrent flights to be dependent on the same revision levels of the operating system and executive system software. The CM subsystem will need to maintain a checklist of software revision dependencies and disallow the download of any software that is not certified to run with the workstation's loaded operating system and executive software revisions.

A primary requirement of the CM subsystem for the support of continuous duration flights will be the ability to integrate new technology into the system over the span of the flight. This will require that the CM subsystem provide for the upload and download of new operating system, executive, and application software to support a new technology. This process should allow for software to be off-loaded onto other machines while the new software or new hardware is installed. The CM subsystem will need to maintain a record of all software which is dependent on a particular revision of hardware or software.

6.3.2 Security Subsystem

Through the use of configuration management and security, the Concept Executive should provide a secure distributed system. The executive shall provide protection during operation through a reliable security subsystem. The security subsystem provided by the executive shall consist of:

- User account security.
- User account administration .
- Limited super user capability.
- Process execution security
- User privilege restrictions.
- Command issuance security.
- Proprietary data protection.

Protection of user accounts shall be provided by UNIX and later by POSIX through the use of permission levels used to allow access to data by users other than the owner. A user's personal account shall be protected during operation through the use of passwords.

Password security shall be provided through scheduled validation. Whenever a new password is entered, an expiration date will be stored. The security subsystem will provide an automated mechanism for flagging passwords which are about to expire.

In order to protect the operating system from access by unauthorized users, the root permission shall be controlled during all modes. Only users with system administration authorization be allowed access to the root privileges. This will prevent unintentional performance impact on a user's activities by another user.

A tracking system will need to be implemented by the security subsystem in order to provide auditing of the activities of system users. The security subsystem shall also provide a means for auditing the activity on any system entity (program, file, etc...). All activities which are performed with root privileges will be audited. The security subsystem shall provide the capability of turning the auditing functions on and off. The security subsystem shall provide a method for reporting the results of an audit.

The executive shall provide a hierarchy of system security controls ranging from individual user permissions to system-wide capability that controls access of groups of users and contains categories of permissions (UNIX-like). This will insure that only authorized users have access to protected files.

The security subsystem shall provide protected access to critical functions and commands. The security subsystem shall not allow potentially harmful commands to be executed during operations. All system commands which will be allowed for a particular mode of operation shall be ac-

cessible through the UNIX operating system shell. The shell itself does not present an environment which is hazardous to the health of the workstation. It is those commands which can be accessed from the shell which may actually cause problems with the workstation's health. A restricted shell based on a special file system shall provide the user with the full flexibility of the regular shell with the exception of those commands which may be used inappropriately.

This subsystem shall provide programmatic and command line interfaces. This subsystem shall not provide an application which summarizes or allows modification of this information.

6.3.3 System State Subsystem

The Concept Executive shall provide interfaces which allow the current state of environment-specific variables to be retrieved and updated. The variables shall include all those necessary to allow proper recognition and control of the state of the environment. The variables provided by the Concept Executive shall include, but not be limited to:

- Current flight.
- List of active flights.
- Host or locally generated GMT.
- Current user and group (flight position).
- Access mode of workstation.
- Access mode of network and hosts.
- Default host for communications.
- Security status.

The System State Subsystem shall provide a set of processes which monitor the state of the system and maintain the listed variables. Additional environment-specific data shall be provided by separate subsystems. This includes the type of data (data acquisition), list of workstations and hosts (directory services), workstation status, etc.

This subsystem shall provide programmatic and command line interfaces. This subsystem shall not provide an application which summarizes or allows modification of this information.

6.3.4 Data Acquisition Subsystem

The Concept Executive shall provide a Data Acquisition Subsystem. The major requirement of this subsystem is to support a uniform set of interfaces to all network data for the application programmer. All network data shall be accessed through these interfaces. This will provide a uniform method for retrieving all data. Network differences shall be transparent to the user.

The Concept Executive shall not allow data loss due to workstation response time. The real-time data acquisition processes shall have the highest priority to ensure this requirement. The system shall be designed such that the maximum burst of data will not result in loss of data at the workstation.

The Concept Executive shall allow applications to access data in an efficient and flexible manner. A mapping of the data shall be supplied to the application program so applications may access the actual data space and not be provided copies of the data.

The Concept Executive shall provide a table driven map of the data to allow data format changes without recompiling source code. This table shall be generated externally of the Concept Execu-

tive. A symbolic reference shall be provided for each data element. This will allow the Concept Executive and applications to be independent of format changes and not require recompilation for each data format change.

The Concept Executive shall allow the storage and access of several sets of real-time data. Each new set shall overwrite the oldest existing set of data. The number of data sets stored shall be controlled by the Concept Executive. A locking mechanism shall be provided within the concept executive to ensure data integrity. This is to prevent a process from using data from two different time intervals.

The Concept Executive shall support several streams of real-time data. Access to these data streams shall be transparent to the applications programmer as much as possible. The concept executive shall support multiple missions. Thus, the real-time data streams may provide data for one mission or multiple missions.

The Concept Executive shall place all applicable data in the display workstation memory. This service is required to support different configurations. Some workstations may not have direct access to the real-time LAN and must have a fast method of accessing real-time data. Interfaces to the application programmer shall not be altered due to the configuration of the LAN.

6.3.5 Event Subsystem

The Concept Executive shall provide an Event Subsystem which queues all event messages generated by the workstation and received from the global network. An event is defined as a string of ASCII text which has intrinsic meaning (as opposed to binary flight data).

The Event Subsystem shall queue and log (under control of user) all received messages. Programmatic and command line interfaces shall be provided to perform the following functions:

- Retrieve (destructively and nondestructively) the last message from queue.
- Retrieve (destructively and nondestructively) the last message of given type.
- Retrieve (destructively and nondestructively) the last message from a given source.
- Enable or disable logging of events to a file.
- Specify the event log file or device.
- Clear the event queue.
- Generate a local event.
- Generate an event for a specific workstation or host.

Each generated event shall have an associated source identifier and type. The Event Subsystem shall allow for unlimited definition of system types and a reasonable set of user-defined event types.

Note that the Event Subsystem shall not provide an interactive application for reviewing generated events. Only simple programmatic and command line interfaces are provided.

6.3.6 General-Purpose Communications Subsystem

The General-Purpose Communications Subsystem provides a consistent set of interfaces to communicate over the system networks. The Concept Executive shall support the following communications:

- Workstations to workstation.

- To and from a host computer.
- Displays between workstations.

The General-Purpose Communications Subsystem shall use ISO COTS packages for the above communications. Thus, to transfer a file between workstations, the application programmer will provide the file name and the destination. The subsystem will use X.500 services to look-up the physical address and use FTAM services to make the connection and transfer the data. Similar services shall be provided for host communications, including commands, and for displays between workstations.

Interfaces shall be developed for network COTS packages to provide uniform access to all functions. The following shall be supported by the General-Purpose Communications Subsystem:

- Broadcast data transfers.
- Multicast data transfers.
- Point-to-point data transfers.

6.3.7 Distributed Processing Subsystem

The Concept Executive shall provide a Distributed Processing Subsystem which provides an integrated set of distributed processing functions for workstations in the network. The major requirements of the Distributed Processing Subsystem are as follows:

- Develop an interface which provides health and status data for the current workstation.
- Support integrated distributed processing and optimum utilization of resources by providing process distribution and load balancing.
- Support fault tolerance for critical user applications.
- Support transparent use of different workstation configurations (configuration independence).

The Distributed Processing Subsystem will use the term "subset" to define a physical or logical collection of workstations. The two types of subsets referenced are as follows:

- Static subsets - a collection of workstations will be divided into static subsets by Configuration Management and/or by physical arrangement of the global networks (such as a local Ethernet network connected via a bridge to the network backbone) . A static subset will most likely correspond to a set of workstations dedicated to support one major application or group of users. Static subsets cannot be modified by application programmers or users.
- Dynamic subsets - within a static subset, users may define dynamic subsets which define another level of workstations groups. The purpose of dynamic subsets is to provide users with the ability to group workstations and treat them as logical systems. This will allow better utilization of available resources.

The Distributed Processing Subsystem shall be fully constrained and controlled by the rules established by Configuration Management. The Distributed Processing Subsystem shall not allow local

subsets of workstations to behave independently of Configuration Management and to have free reign of local networks.

The Distributed Processing Subsystem can only be as effective as allowed by the bandwidth of the network. If the networks are so overloaded that it is impossible to transfer data, then this subsystem will not be effective.

In order to distribute processing loads and provide the defined form of fault tolerance, the system shall be able to provide an identical operating environment on each workstation in a dynamic subset. This is necessary, as it is not useful to distribute a process which provides results due to a different configuration. This shall be accomplished via the Data Acquisition Subsystem and possibly a remote file system.

The functions provided by the Distributed Processing Subsystem shall only be available for workstations in the same mode of operation.

The following sections discuss the primary goals of distributed processing support and address specific requirements for each.

6.3.7.1 Health and Status Information

The Distributed Processing Subsystem shall obtain the required local health and status data for use in load balancing, fault tolerance, and for routing to a global health and status application. The Distributed Processing Subsystem shall periodically obtain the required statistics and place them in a structure which is available to all local processes. The data values to be obtained shall include, but not be limited to the following:

- CPU percentage for:
  - User.
  - System.

  Percentages will be provided for all local CPU's (multi-processor systems).
- Memory utilization including:
  - Used and available memory pages.
  - Memory pages paged in and out
  - Memory pages swapped in and out.
- Disk operations on all disk controllers and drives.
- System interrupts, system calls, and context switches.
- Process-specific information:
  - Number of active processes.
  - nNmber of waiting processes.
- Graphics activity (load on graphics processor if available).

Health and Status information retrieval is part of the Concept Executive as this data is required for support of the Distributed Processing Subsystem. The collection of such information will also be machine dependent and therefore should be isolated within the Concept Executive.

If the local health and status information is required by another workstation or host, the interested system shall be responsible for initiating the actual network transmission.

6.3.7.2 Process Distribution and Load Balancing

The Distributed Processing Subsystem shall support distribution of processes in order to balance the processing load of the workstations within a dynamic subset. The subset load is defined as the aggregate of the following data values for all workstations in the subset:

- CPU percentage.
- Memory utilization.
- Disk operations.
- Graphics operations.

The load balancing function shall take these and any other relevant data values into consideration when distributing processes. The important point is that the load of the subset is more than merely the current utilization of CPU's. The system could have a very low CPU utilization, but still be overloaded due to heavy use of another resources.

The Distributed Processing Subsystem shall provide an interface which allows execution of an application on a workstation in the dynamic subset based on the current load. This interface shall examine the load of all subset workstations and execute the application where the least load exists (or as defined by the loading algorithm). This interface shall also allow specification of how output data is returned (via file system or display).

The Distributed Processing Subsystem shall provide an interface which allows a workstation to control its participation in load balancing. An interface shall be provided to insure that all applications are executed locally independent of whether or not other workstations are available for load balancing.

The Distributed Processing Subsystem shall provide an interface which allows a workstation to provide its resources for use in load balancing. The ultimate control of whether or not a workstation participates in load balancing remains with the individual workstation. This interface shall allow a workstation to be immune from requests from other workstations for resources. To provide resources for load balancing, the workstation may specify access to all workstations in the static subset or to selected workstations (thereby establishing a set of dynamic subsets). Workstations already participating in load balancing shall automatically sense the changes in the subset. This approach follows the classic server-client model.

The Distributed Processing Subsystem shall adopt a coherent mechanism and policy for treatment of applications running on workstations which have been disabled. This mechanism may be to wait for the application to complete or terminate the application in an orderly manner.

The Distributed Processing Subsystem shall provide an interface which allows the user to select and control the manner in which load balancing is performed. The algorithm shall be used for all workstations in the static subset. The attributes available for specification shall include the following:

- Balancing algorithm.
- Weight given to CPU, disk, memory and other resources.

6.3.7.3 Application Fault Tolerance

The Distributed Processing Subsystem shall provide fault tolerance for critical applications. The purpose is to define a critical application, sense its failure (due to software or hardware), and then

restart the application on the current or another workstation. This is a very simplistic form of fault tolerance which is a logical extension of the functionality in process distribution. True fault tolerance would involve redundant operations and/or the ability to warm start (point after failure) applications. The proposed form of fault tolerance shall only provide cold restart of applications.

The Distributed Processing Subsystem shall provide an interface which allows identification and execution of a critical application. An additional interface shall be provided to change the status of an already running application to critical status. It shall also be possible to automatically determine that an application is critical by some data store in the executable file.

The Distributed Processing Subsystem shall retain a record of critical applications running in the current dynamic subset. This information is distributed to allow restart of applications after failure of an entire workstation. The Distributed Processing Subsystem shall sense the failure of a critical application and take the appropriate action. An interface shall be provided which allows specification of the action to take upon critical application failure. The options include the following:

- Re-start the application on a local workstation or remote workstation if necessary.
- Re-start the application on the workstation with lowest load.

In event of a failure, the Distributed Processing Subsystem shall generate and route events to all workstations in the appropriate dynamic subset.

6.3.7.4 Configuration Independence

The Distributed Processing Subsystem shall support several configurations of workstations (configuration independence). The goal of configuration independence is to identify and support additional configurations which satisfy the real-time requirements of the environment and share the various hardware resources of the workstations. The Distributed Processing Subsystem shall support such configurations and mask the configuration-specific details from both other Concept Executive subsystems and application programmers.

One of the major requirements in a real-time data driven system is to expedite the delivery of data to all application processors (workstations). One solution is to directly connect each workstation to the global networks. This configuration solves the problem of data distribution, but requires each workstation to be configured with a large number of resources (network boards, disks, tape, etc.) to communicate with the network and function independently. Assuming that a workstation has a single graphics display, this configuration must be duplicated for each processing node in the network.

A solution provided by a few vendors is to support multiple displays off of each workstation processor. By attaching 2 or more displays to a workstation, expensive hardware resources and interfaces are shared and a significant cost-per display advantage is realized without jeopardizing delivery of real-time data. Figure 6-4 illustrates this configuration.

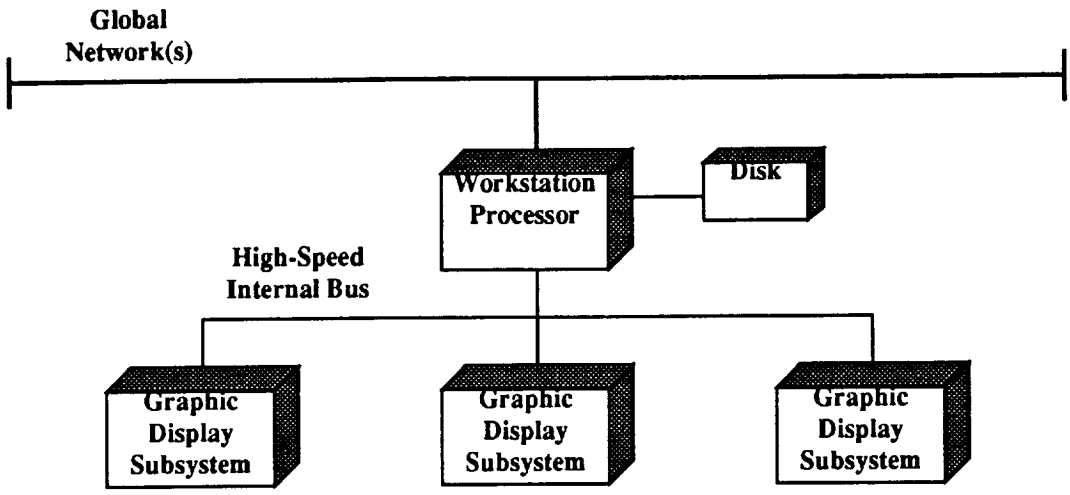


Figure 6-4 Workstation Configuration with Graphics Displays

A problem with this configuration is that it is not adequately supported by a large number of workstation vendors. In addition, placing too many graphics displays on a workstation can saturate the processor and peripherals. Such a configuration is also not as fault tolerant as single display workstations, as if the workstation processor fails, all connected graphics displays will fail as well.

The following two subsections discuss alternative configurations which shall be supported by the Distributed Processing Subsystem.

6.3.7.4.1 X Terminals As Displays

One promising technology is that of X terminals. An X terminal is a device which provides a keyboard, color or monochrome bitmapped display, and an X server. The X terminal does not provide any local processing capability, but rather is connected to the network and depends on a host to execute applications (X Windows clients). The clients execute and route graphics in the X protocol to the X terminal via the network. The X terminal interprets the X protocol and generates the appropriate graphics display. As the X protocol is standard, any X terminal meeting performance requirements (color, display size, etc.) may be used for graphics generation.

A workstation configuration utilizing X terminals as graphics displays is presented in Figure 6-5.

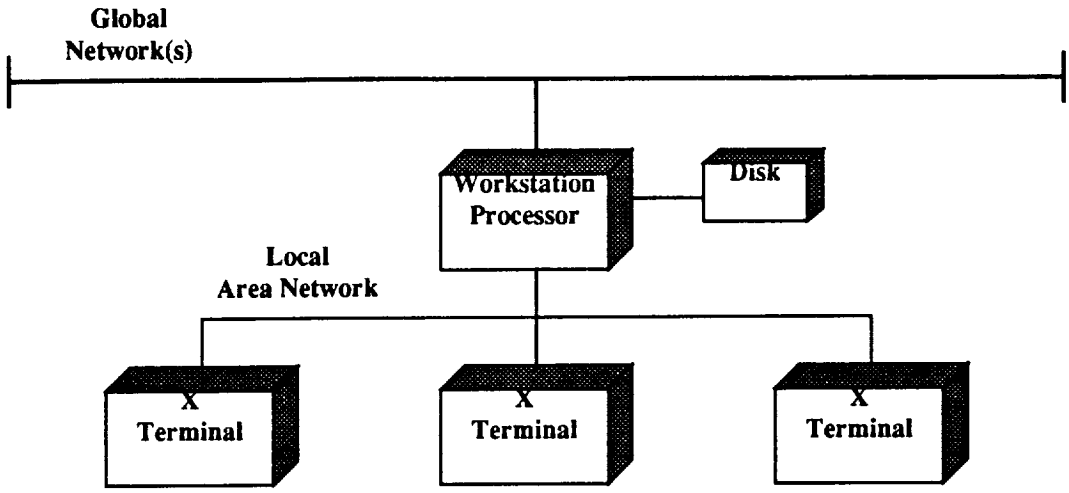


Figure 6-5 Workstation Configuration with X Terminals

Although a relatively new technology, there are a number of high-performance X terminals currently available. These devices provide custom hardware which allow excellent graphics performance. Use of X terminals off-loads much of the graphics processing which is normally performed by the workstation processor. Another advantage is that there are a number of vendors marketing these devices. This competition will keep the cost of X terminals low and insure rapid introduction of new technology. The primary disadvantage of X terminals is that they are limited by the speed of the local area network.

From a software perspective, a configuration using X terminals is very similar to one using local graphics displays, as all processing still resides on the workstation processor. The Distributed Processing Subsystem could easily support the two configurations.

The Distributed Processing Subsystem will not be able to specify the network protocols used to communicate with the X terminals. Therefore to support X terminals, the Distributed Processing Subsystem may have to use another protocol on the local network serving the X terminals. While this is undesirable, it does not introduce a configuration management problem. In the future, X terminals will allow communication via several protocols and should no longer have this problem.

6.3.7.4.2 Separate Processors as Displays

A totally different configuration is one in which several actual workstations with separate processors are used as displays. In this configuration, a server workstation is attached to the global networks. The server in turn provides data and or disk services to the actual display workstations. Figure 6-6 illustrates this configuration.

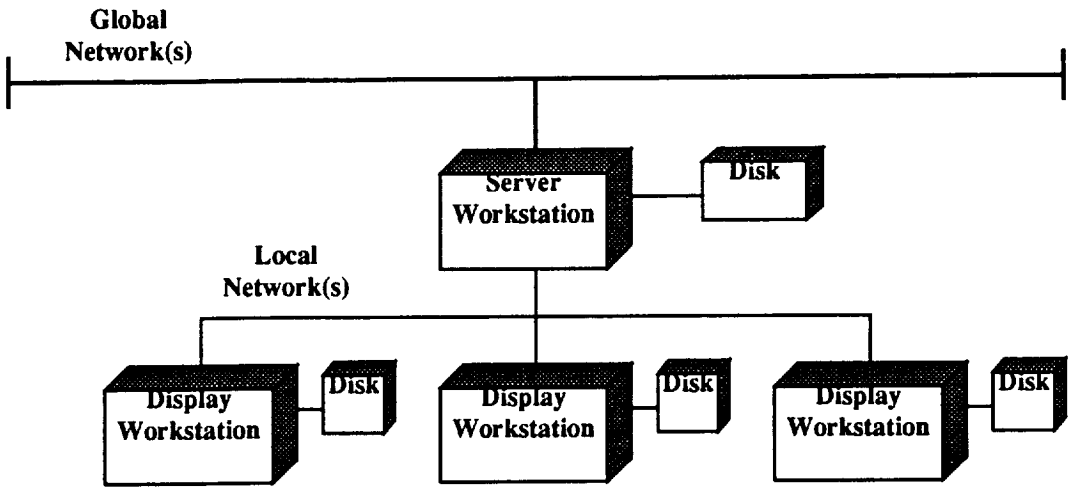


Figure 6-6 Server/Client Workstation Configuration #1

In this configuration, the server workstation will provide all data and some disk services to the display workstations. The small local disks are optional, but recommended to provide local swap and other services to reduce local area network traffic. This configuration may use one or two local area networks to allow adequate bandwidth for disk and data support. This may be necessary as a large amount of data will pass through the server and on to the local network.

A similar configuration is one in which the display workstations are directly connected to the global networks in order to retrieve real-time data. Other data and disk services are provided via the server workstation. Figure 6-7 illustrates this configuration.

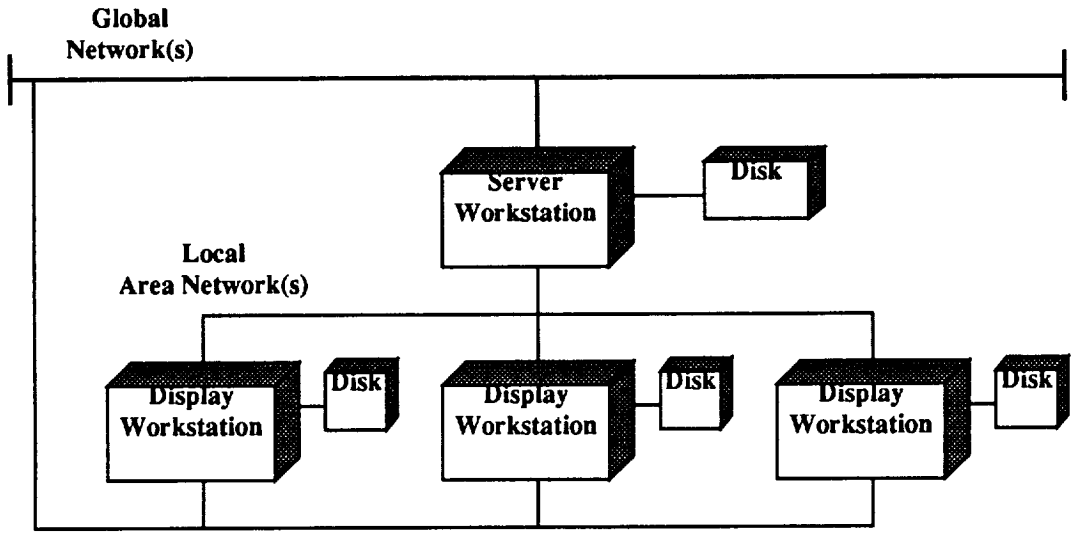


Figure 6-7 Server/Client Workstation Configuration #2

There are a number of distinct advantages to this type of configuration. This includes the following:

- Use of workstations in this manner is widely supported by vendors. The workstations used for server and display functions are interoperable.
- Use of individual workstations provides more total processing capability.
- Individual workstations are often less expensive (and always more flexible) than dedicated graphics displays.
- Workstations can share use of the server disk. This will also reduce the number of copies of software for which the configuration must be managed.

There are also a number of disadvantages, including distribution of data, real-time performance, and software complexity. These disadvantages are offset by the advantages of supporting this type of configuration.

6.3.8 User Interface Subsystem

The Concept Executive shall provide a User Interface Subsystem which presents an environment which is consistent across all operating modes (subject to configuration management and security). This means that regardless of which mode of operation a workstation is executing under, the interface to the Concept Executive and the operating system shall remain essentially the same. The only difference shall be in the restrictions placed on the commands which are allowed to be executed during different modes.

The User Interface Subsystem shall provide the following three COTS and custom-based functions:

- Provide a login application which allows controlled entry to the Concept Executive system.
- Establish a graphic user interface-based interactive environment with a consistent look and feel.
- Provide a real-time mechanism for efficient and deterministic display of text.

Each of these functions will be described in more detail in the following three subsections.

6.3.8.1 Login Client

The Configuration Management Subsystem of the Concept Executive requires a new level of control for user access to the system. The existing login and initialization process of UNIX is not sufficient to achieve the required level of control. The User Interface Subsystem shall support configuration management by presenting a new login client which completely replaces the existing UNIX process. This client shall present an easy-to-use interface and shall allow the following:

- Entry of information which identifies the user (username, password, group). Note that the password shall not echo when entered.
- Present environment-specific information and allow user to select values which affect initialization of the session (flight, host, network, etc.).
- Specify the access mode (development, simulation, operations) and default values to use (CM workstation, CM host).
- Specification of files which allow special environment initialization.

When the user attempts a login, all entered information shall be verified. If attempting entry to operations mode, the username and password shall be verified by configuration management. If access is allowed, the environment shall be initialized as required by configuration management and as specified by any available user initialization files.

The login client shall be active on every graphics display on each workstation controlled by the Concept Executive. The login client shall be reactivated after a user session is terminated.

The login client is the only custom graphic user interface application provided by the User Interface Subsystem. This client is required to replace the existing UNIX process and to provide a logical manner in which the COTS-based interactive graphic user interface is initiated. The login client shall provide an interface which conforms to the look and feel specification provided by Motif.

The login client shall therefore act as a guide for development of other graphic user interface applications. The only window-based client provided by the Concept Executive shall be the login client. This client shall provide the standard look and feel of the system. Once a user has logged into a workstation session, the Concept Executive shall place the user into a COTS window environment. This environment shall include xterm, the selected window manager, and access to the selected shell.

6.3.8.2 User Environment

The Concept Executive shall provide a simple interactive environment from which all types of users can effectively interact. This environment depends on a collection of COTS clients as described in the "Graphic User Interface" section. This environment shall allow execution of commands via the native command line interpreter (the UNIX shell). Interaction with the shell is the primary means of executing commands in this environment.

Through the selection of the Motif widgets, the User Interface Language, and the associated look and feel, the Concept Executive shall mandate a consistent graphic user interface behavior. The Concept Executive shall provide a standard set of interaction constructs to applications programmers. This set of constructs will then help to maintain the consistent look and feel throughout all applications developed in the system. The types of interaction constructs which are standardized by this look and feel shall include, but not be limited to the following:

- Colors.
- Fonts.
- Function keys.
- Mouse inputs.
- Cursor image.
- Keyboard manipulation of the graphic cursor.
- Use of menus, scrollbars, and other user interface objects.

The Concept Executive shall rely on a COTS window manager ("mwm" as previously described). Only one window manager shall be supported in order to maintain a consistent look and feel. This is necessary, as in addition to presentation of a different interactive interface, different window managers require development of separate code to properly interact with each window manager.

6.3.8.3 Real-time Display Mechanism

The User Interface Subsystem shall provide a programmatic interface which allows real-time display of large quantities of text values. This interface shall allow limited use of colors, fonts, and other simple attributes. The purpose of this interface is to provide a standard mechanism for displaying data in an efficient, deterministic manner. The purpose is not to provide a general purpose mechanism which supports a wide variety of text attributes and other graphics.

A requirement of the Concept Executive is to allow real-time acquisition and display of telemetry data. This is a problem in an X Windows environment for the following reasons:

- X Windows is asynchronous - the server queues graphics requests and flushes internal buffers in a non-deterministic manner.
- X Windows does not provide a simple and efficient manner of displaying large quantities of text values. Users must either use cumbersome low-level calls or use a widget which is not well suited to the requirement.

The User Interface Subsystem shall utilize the real-time features of the operating system, the lowest level X Windows interface, and if necessary any proprietary X extensions to provide an elegant mechanism for presenting large quantities of text values. This interface shall provide the following functions:

- Allows specification of individual or a buffer of values to be used for update.
- Only updates values which have changed.
- Allows specification of display action taken for portions of the window not exposed (obscured or iconified - this is important, as some X servers do a poor job of handling this case).
- Allows specification of colors and fonts to be used as defaults and for limit conditions.

7.0 Technology Survey Introduction

The research direction of the first phase of NASA Grant NAG 9-340 is to survey government and commercial sites for systems and technology applicable to workstation executives. A workstation executive is a software system which integrates a network of workstations and provides a standardized set of services to application programmers. This document defines a "Concept Executive" which integrates the native operating system, additional Commercial-Off The-Shelf (COTS) standard interfaces, and custom subsystems in order to provide the functionality required by a real-time spacecraft telemetry processing command and control environment. For more information on the Concept Executive, refer to chapters 1 through 6.

The Technology survey was conducted to identify software technology useful in the Concept Executive. The two major types of software surveyed include the following:

- Identify operational (or planned) executives from which technology and design approaches may be identified and used in the Concept Executive described by this document.
- Identify related software systems which are applicable to the Concept Executive. This effort consisted of the review of two user interface management systems.
- Identify and describe in detail, new standards which are applicable to the Concept Executive.

The scope of this survey effort covers software which relates to the Concept Executive. The technology survey does not include reviews of hardware components or miscellaneous COTS software which is not a standard.

For each system or standard reviewed, the following information will be provided (the level of detail and content will vary based on the type and complexity of the system):

- Introduction - (all) a brief description of the history and purpose of the system.
- Contact point - (all) a listing of the organization or individual from which the system or information may be obtained.
- Review process - (executive and user interface systems) a brief description of the process used to review the system. This will consist of reviewing documentation and/or experimenting with code. A list of all reviewed documents is provided for the convenience of the reader.
- System Requirements - (executive systems) if available, the high-level requirements for the system will be presented. Requirements are presented in detail, as the final result of this grant will be expressed in terms of requirements.
- Description - (all) a more detailed description of the system or standard.
- Applicability to the Concept Executive - (executive and user interface systems) an overview of what technology is useful and how it may be applied to the Concept Executive.

Note that documents referenced in the "Review Process" sections will not be repeated in the document Bibliography.

8.0 Executive Systems

This chapter summarizes several actual operational or planned systems used at various NASA installations. The systems reviewed (and the installation where used) include the following:

- Transportable Payload Operations Control Center (TPOCC) - Goddard Space Flight Center.
- Generic Checkout System (GCS) - Kennedy Space Center.
- Multi-satellite Support Operations Control Center (MSOCC) Application Executive (MAE) - Goddard Space Flight Center.
- Peripheral Processor System (PPS) - Marshall Space Flight Center.
- Image Reduction and Analysis Facility (IRAF) - National Optical Astronomy Observatories.
- Trajectory Operations Application Support Task (TOAST) - Johnson Space Center.
- Space Flight Operations Center (SFOC) - Jet Propulsion Laboratory.

Each system was obtained by contacting the appropriate individual(s) at the different NASA centers. The review processes consisted of on-site demonstrations, documentation reviews, and experimentation with actual code. The following sections describe each system in more detail.

8.1 TPOCC System

The Transportable Payload Operations Control Center is a system undergoing development at Goddard Space Flight Center. In SwRI's understanding, Goddard is responsible for the majority of unmanned orbiting satellite projects. Each satellite has its own requirements and will contain a unique collection of instruments. Although many satellites are similar (such as Multi-Mission satellites), there remains a wide range of unique ground support requirements which must be met for each.

The purpose of the TPOCC system is to provide a standard hardware and software concept which may be easily configured to the requirements of each satellite. It is intended to replace the existing system which must be reconfigured for each supported satellite. This system includes specialized hardware and software for each such satellite. This is wasteful as a large amount of development is required for each satellite. Specific maintenance costs and expertise are also required for each such system.

For each new satellite, existing standard TPOCC software will be configured and then transported to the new system. The TPOCC system provides the generic functions and provides mechanisms whereby satellite-specific requirements are implemented. In this way, development costs are minimized and the time required to ramp up for a satellite project is greatly reduced. The TPOCC concept defines a generic and flexible system which can be reduced or expanded to suit the requirements of a given satellite (or any spacecraft).

As of the date of this review, the TPOCC was in the form of a prototype called the Prototype Transportable Payload Operations Control Center (PTPOCC). The purpose of the PTPOCC is to demonstrate the feasibility of controlling NASA satellites with a system based on the TPOCC concept. The PTPOCC will be used to support the Solar Maximum Mission (SMM) satellite. This project (called the SMMOCC) was selected for the following reasons:

- The feasibility of the TPOCC concept must be verified before used on a new operational system.
- The SMM is a standard Multi-Mission Satellite (MMS). This will allow the software to be easily used on other satellites of this type.
- The SMMOCC is nearing the end of its life cycle and will need to be replaced if SMM is refurbished.

The PTPOCC system will be used in conjunction with the existing SMMOCC system to demonstrate the feasibility of the TPOCC concept. This is of course similar to the role of the Transition Flight Control Room (TFCR) at NASA-JSC. In addition to its basic functionality, the PTPOCC will include SMMOCC-specific functions to support the unique requirements of the SMM satellite.

For the remainder of this document, the term "TPOCC" will be used to describe the basic system concept. The term "PTPOCC" will be used to describe the prototype implementation.

8.1.1 Contact Point

Code 511

Goddard Space Flight Center

8.1.2 Review Process

The review process for the TPOCC system involved examination of the following two documents:

- Prototype Transportable Payload Operations Control Center (PTPOCC) System Design Review (SDR) Book 1 of 2.
- Prototype Transportable Payload Operations Control Center (PTPOCC) Preliminary Design Document.

The documents reviewed provided a good overview of the TPOCC system and a detailed discussion of the software design. The documentation was well-suited for this review.

8.1.3 TPOCC Requirements

The basic TPOCC system, as it applies to any application of its concept is summarized in the following primary requirements:

- The TPOCC must be portable, expandable, configurable software plus a host computer system for the operation of spacecraft and/or instrument payloads. TPOCC will be portable in the sense that the hardware and software will be usable for a variety of satellite projects.
  - The hardware used for the TPOCC will be compact and must not require special housing or environments.
  - TPOCC will be expandable in that the software, processors, networks will be integrated as needed for a particular satellite application. The minimum TPOCC system will be a single computer; the maximum will be a networked set of computers.
- The TPOCC system will include three levels of software libraries. These include the following:
  - Kernel Library - contains routines which perform generic system and application-independent functions.
  - System Specific Library - contains routines which provide a standard interface to a variety of host computer systems.

  - User Extensions Library - contains routines implementing functions specific to a particular satellite project.
- The TPOCC will provide a system generation facility to allow subsets to be generated.
- The TPOCC software will perform functions common to a wide range of spacecraft and instrument applications. This includes operator command, display, telemetry processing, high-rate dump collection, and real-time command processing.
- All TPOCC software shall be written in Ada to allow porting to any system supplying a DOD certified compiler. It must be modular and allow user extensions to be easily added.
- All TPOCC software shall be designed to take advantage of modern host computer and software capabilities. Support for networked workstations, commercial databases, and operator interaction through different devices are required.
- The TPOCC will use commercially available hardware and software to minimize costs. Such products will be selected based on performance and the requirements of a particular satellite.

For the PTPOCC system, the primary requirements are expanded to address the subset of functions provided in the prototype system. These requirements are further divided into functional and constraining requirements. The following sections summarize these two types of requirements.

8.1.3.1 PTPOCC Functional Requirements

PTPOCC functional requirements describe the actual functions which must be provided. These requirements represent the subset of TPOCC functional requirements which will be implemented in the PTPOCC. It does not include the project-specific functions added to support the SMMOCC. Each of the following sections describe in detail one area of the functional requirements.

8.1.3.1.1 Generic Functions

Generic functions include those which affect the entire system, including processing of operator commands, display processing, and handling of system events. The generic functional requirements are summarized below:

- Operator command processing - a TPOCC Operations Language (TOL) shall be provided as a means of initiating functions:
  - All real-time functions shall be controllable via TOL statements.
  - TOL statements shall be read from keyboard, procedure, or via screen objects (such as menus).
  - PTPOCC shall parse and execute TOL statements.
  - It shall be possible to issue TOL statements on one processor in order to control functions on another processor.
- Operator display processing - display functionality will be provided to allow users to review data and control the system:
  - All real-time functions shall be monitored via operator displays.
  - Displays shall be generated in windows which are network-transparent.
  - Support for overlapped and tiled windows shall be provided.
  - Text and graphic display of data shall be supported.
  - Text shall be static or dynamic with color support.

- Engineering conversion shall take place for display purposes.
- PTPOCC shall be able to generate state name displays of discrete telemetry points.
- Graphic displays shall consist of dynamic 2-D color plots (strip-charts and x-y plots).
- Cascading (hierarchal) menus shall be supported.
- Display definitions may be defined, saved, and used.
- Display snapshots may be generated for hard-copy output.

• Event processing - PTPOCC will be capable of processing different types of events:

- Each PTPOCC function shall detect events and generate event messages.
- Each event message shall be logged by the detecting processor.
- Each event message shall have an associated event number and class.
- Each event message shall be distributed to each display which has indicated interest in the event's number and/or class.
- Events occurring at a high rate of speed will not interfere with PTPOCC real-time processing.

8.1.3.1.2 Real-time Functions

PTPOCC shall support real-time control and monitoring functions. Specifically this includes real-time telemetry and command processing, performance monitoring, and test data generation. These requirements are summarized below:

• Real-time telemetry processing:

- PTPOCC shall read the MMS format real-time telemetry and associated communications data.
- All real-time telemetry and associated communications data shall be Greenwich Mean Time (GMT)-tagged and saved on disk.
- PTPOCC shall provide playback for telemetry and communications data at a specified rate. It will not be necessary to perform playback and real-time concurrently.
- Monitor frames of telemetry shall be decommutated. The most recently received value for each telemetry point shall be stored in a table and made available.
- Selected telemetry points shall be streamed out for graphic displays.
- Trend-checking shall take place as decommutation is performed.
- Limits checking shall be performed during decommutation.
- Spacecraft configuration checking shall be performed on major frames.
- Limits checking may be turned on/off for each telemetry point.
- PTPOCC shall execute user-defined functions on receipt of a major frame of telemetry, receipt of a minor frame of telemetry, or when a specified point is updated.
- Real-time telemetry processing shall be data-base driven.

• Real-time command generation:

- PTPOCC shall generate MMS-format real-time commands in response to TOL statements.
- PTPOCC shall compute and append the error detection code for each real-time command and produce real-time command frames.
- Command block output shall be metered to the selected MMS commanding rate.
- TOL real-time command generation control statements shall only be processed from one workstation at a time.
- All transmitted real-time commands shall be GMT-tagged and stored in the command history file on disk.

- Real-time command generation shall be data-base driven.

- Real-time command verification:
  - The spacecraft command counters shall be monitored for expected values.
  - For each real-time command issued, a list of telemetry points shall be monitored for the expected raw values until the points and values match or the verification times out.
- Performance Monitoring - The PTPOCC will be capable of periodically computing and presenting system performance measurements. The statistics monitored include:
  - CPU utilization.
  - Ethernet and disk traffic.
  - Memory utilization.
  - Context switching.
  - Process state.
  - Process priority.
  - Real and virtual process size.
  - Process CPU utilization.
- Test Data Generation - PTPOCC will generate data to be used for software testing purposes:
  - PTPOCC shall generate minor frames of text telemetry from a canned major frame.
  - PTPOCC shall generate associated communications data.
  - PTPOCC shall modify telemetry in response to real-time spacecraft commands.
  - PTPOCC shall modify telemetry in response to TOL commands.

8.1.3.1.3 Off-line Functions

PTPOCC shall be able to generate flat files from the PTPOCC database for real-time support and for report generation. The specific requirements are listed below:

- Flat file build:
  - PTPOCC shall generate flat files from the PTPOCC database.
  - PTPOCC shall generate flat files from display definitions and the PTPOCC database for use during real-time.
- Reports:
  - PTPOCC shall be able to generate reports on the database contents.
  - Reports shall be generated to show portions of the telemetry history file.
  - Reports shall be generated showing selected portions of the command history file.
  - Reports shall be generate showing portions of the event history file.

8.1.3.2 Constraining Requirements

Constraining requirements include those which dictate directions of the PTPOCC design and implementation. These requirements are of interest again due to their applicability to the Concept Executive. The constraining requirements are summarized in the following sections.

8.1.3.2.1 System Constraints

The PTPOCC system shall meet requirements of the host hardware and software as described below:

- Host Computer Hardware:
  - PTPOCC shall be implemented so that it can run on one processor, multiple processors communicating over a system back-plane, or over multiple processors connected by Ethernet..
- Host Computer Software:
  - Operating System - a widely available system such as UNIX.
  - Language - C and/or Ada.
  - Data Base - commercially available relational DBMS.
  - Communications/Networking - TCP/IP.
- Graphics:
  - Support for windows, icons, menus (the basic desktop model).
  - A network window system is required to generate windows which may be distributed over the network (client/server).
  - Off-the-shelf plotting package for graphics and text display.

8.1.3.2.2 Interface Constraints

The PTPOCC system shall support all existing communication and user interfaces as described below:

- NASA Communications (NASCOM) communications interface.
- User interface - The PTPOCC shall use modern workstations supporting the following:
  - High-resolution color displays supporting windowed text and graphics.
  - Color printers for full screen output.
  - Laser printers for monochrome screen output and listings.
  - Input via keyboard and pointing device (mouse).
- GMT interface.

8.1.3.2.3 Performance constraints

The PTPOCC must be able to sustain processing of up to 64Kbps of telemetry without data loss.

8.1.3.2.4 Life Cycle Constraints

PTPOCC shall have the design qualities and meet the development, maintenance, and enhancement constraints presented below:

- Maintainability - COTS components (hardware and software) shall be used where possible to minimize development costs.
- Enhancability:
  - PTPOCC software shall be data-base driven.
  - User hooks will be provided for specific processing of real-time telemetry.
- Portability - PTPOCC shall be developed in C and/or Ada and developed in a standard operating system such as UNIX.
- Modularity/Portability - PTPOCC shall consist of three levels of libraries:
  - Kernel - contains portable functions implementing the standard TPOCC functions.
  - System-Specific functions - contains functions which interface the kernel library to the host computer environment.
  - User extensions - contains spacecraft-specific functions.

8.1.4 System Description

The TPOCC system will be a flexible and portable system which will provide the foundation for support of a wide variety of satellite (or more generically "spacecraft") projects. The TPOCC concept is based on a completely open architecture which depends on a number of industry standards. This is necessary to achieve the flexibility, portability, and modularity requirements. The TPOCC concept depends heavily on standard network hardware and software for distributed processing. It is this distribution which allows the system to be reduced or expanded based on satellite requirements. Processing capability is increased by adding additional processing nodes to the standard network.

In addition to network standards, the TPOCC system is based on graphic and operating system standards. While the concept describes a level of library in which system dependencies are implemented, it is intended that this library primarily consist of standard functions. This greatly reduces the amount of system-specific software which must be developed.

The TPOCC system achieves many of its goals by designing software subsystems which can operate together on the same processor or across a network of homogenous or heterogeneous processors. In this way, subsystems may reside on processors best suited to their requirements. For example, real-time telemetry acquisition resides on a real-time processor (which does not require high-resolution graphics). Operator interface subsystems reside on a graphics workstation (which does not require a real-time operating system). This application of distributed processing allows a wide variety of cost and performance effective hardware configurations.

In addition to working in a network of processors, the TPOCC concept will function on a single processor system. This will require a system with both real-time abilities, workstation graphics, and all required support software. While there are systems which meet all requirements, they are not cost effective compared to a distributed system. This of course may change in the future as more UNIX systems provide real-time capabilities.

The remainder of this section discusses the TPOCC system in more detail. This discussion is divided into the following three subsections:

- Commercial Technology Review - a summarization of hardware and software technology reviewed for use in the PTPOCC.
- Software System Design - a summary of the PTPOCC software subsystem design.
- Implementation Plans - a summary of the current state of the PTPOCC and future plans for prototype evaluation.

8.1.4.1 PTPOCC Commercial Technology Survey

In order to find hardware and software which would meet the constraining requirements of the PTPOCC, a survey of commercial technology was performed. This survey was similar to the standards evaluation performed by SwRI, but also included a close look at various types of hardware and some non-standard software systems. Although beyond the scope of this document to examine this survey in detail, it is useful to mention some of the different technologies which were recommended by this evaluation. The evaluation reviewed technology in the following two areas:

- Standard networks.
- Network nodes (workstations).

Each of the two areas includes an evaluation of hardware and software. The technologies recommended are presented in the following subsections.

8.1.4.1.1 Standard Networks

The central concept in the PTPOCC system is to build the system around a standard network. This allows processors to be added as needed to meet the processing requirements of the system. Using a standard network and communications, software can easily communicate with systems on separate processors.

The standard network functions selected for the PTPOCC system include the following hardware and software:

- Ethernet hardware and protocols.
- Transmission Control Protocol (TCP) and Internet Protocol (IP).
- Universal Datagram Protocols (UDP).
- Remote Procedure Calls (RPC) and EXternal Data Representation (XDR).
- Network File System (NFS).

This set of standards is the basic layer structure present on many popular UNIX systems. In addition to standard networks, the evaluation also reviewed the following network-transparent window systems:

- X Windows.
- Network Extensible Window System (NEWS).

The X Windows system was selected for the PTPOCC system. Note that due to the lack of commercially supported X Windows systems, other proprietary systems were used as interim solutions.

8.1.4.1.2 Network Nodes (Processors)

Two basic types of network nodes were evaluated. These include workstations and real-time processing nodes. Note that there is an overlap area as some workstations have real-time capability. A number of popular workstations were evaluated, including:

- Sun 3 and 4 systems .
- IBM RT system.
- Hewlett Packard 300 system.
- MASSCOMP 6600 system.

In general, each system supported the required network software. Only the IBM and HP supported X Windows and only the HP and MASSCOMP supported real-time functionality. The findings indicate that an HP or MASSCOMP could likely be used if all software were to be placed in one processor. Due to the relative price of these systems, it was judged that this approach was not cost-effective. The Sun and the RT were judged to be good for operator display stations, with the IBM RT considered preferable due to availability of X Windows and better user response during graphics output.

It is important to note that the evaluation was performed in early 1988. It is also somewhat small in scope considering the large number of available workstations. Such evaluations need to be updated at least on a yearly basis to insure that information is up to date.

The evaluation also looked at dedicated real-time systems. Such systems would be used as front-ends to acquire the real-time telemetry data. This data could then be distributed to the processing nodes via Ethernet. For a relatively small environment such as the PTPOCC, this approach is preferable as it allows non-real-time processing nodes to be added as needed. Using a dedicated real-time system is also preferable to buying a real-time UNIX system as a premium is paid for its unique capabilities.

Two separate dedicated real-time processing systems were examined. The systems include the following:

- VAXELN.
- VxWorks.

VAXELN is software which runs on a DEC VAX system. Its major shortcoming is that it communicates via DECNET protocol rather than TCP/IP. VxWorks runs on a variety of 680x0 VME systems, including Sun's. In general, VxWorks was considered preferable due to is speed (relative to processor), use of standards, and foundation in UNIX. Another advantage is that there are a large number of inexpensive 680x0 VME-based systems. This makes it possible to select hardware which is ideally suited to the processing requirements of the system.

8.1.4.2 PTPOCC System Design

The central concept behind the PTPOCC system is a modular design which is flexible, portable, and expandable. The concept assumes one to many separate processors connected on a standard network. The system design places no constraints on the types of hardware or operating system software, as long as a standard means is provided for inter-network communications.

The PTPOCC system is divided up into 8 software subsystems. These subsystems can all reside on one processor or can be divided up and placed on different processors. All subsystems communicate with one another via standard TCP/IP-based sockets. This communications scheme allows transparent access on a single processor or over an Ethernet. The major advantage to this design is that processors can be added as requirements dictate. Subsystems can also be placed on hardware which best suits the subsystem requirements. For example, real-time data acquisition subsystems on real-time processors and operator interface subsystems on workstations.

The PTPOCC system is to be implemented using industry standards as described in the previous section. This is of critical importance to the software as it is necessary for software subsystems to communicate with one another. Use of standards is also necessary to insure that different workstations and other hardware can be interchanged as necessary.

The PTPOCC system will be implemented in the C language to insure portability. Although use of Ada is a specific requirement, it is not yet well integrated into any of the workstation environments and would be difficult to use.

8.1.4.2.1 PTPOCC Software Subsystems

The PTPOCC software system is partitioned into 8 software subsystems, each of which provides a well-defined subset of functionality. The 8 subsystems are as follows:

- Operator Command.
- Operator Display.
- Events.
- GMT.
- Spacecraft Communications.
- Spacecraft Telemetry and Command.
- Spacecraft Communications Simulation.
- Spacecraft Telemetry and Command Simulation.

The following 8 subsections provide brief descriptions of each software subsystem. Each subsystem will be discussed in terms of the processes which it defines.

8.1.4.2.2 Operator Command Subsystem

The Operator Command Subsystem is responsible for reading operator commands from normal input devices and procedure files, parsing the commands, and if necessary, distributing the commands to the appropriate processor for execution. The basic design allows control of both local and remote functions. This is necessary as real-time functions may (and most likely will) be present on a remote processor. This subsystem consists of the following processes:

- Main parser - this process is the central point for entry of commands. The main parser reads the command, parses it, and if legal, sends it to the appropriate process for execution.
- Procedure parser - this process is used to execute commands from a procedure file (as opposed to operator input).
- Router/Server - this process aids the communication of commands to remote processors. This process initiates a communications link to the remote processor.
- Router - this process maintains a single connection to a remote node. Locally, it sends the command to the remote node; the remote router receives the command and passes it to the remote main parser.

8.1.4.2.3 Operator Display Subsystem

The Operator Display Subsystem is responsible for retrieving data, formatting the data into textual and graphical formats based on a display definition, and then presenting this display to the user.

The display generated is network transparent, meaning that the generation may occur on one processor, while the presentation and interaction occurs on another. This is the major motivation for selection of a network-transparent window system such as X. The processes which make up this subsystem include:

- X server - this process renders the display. It may be on the same processor or on a remote processor in the network (note that this is the COTS X server).
- Display - this process builds the display based on the display definition. It receives data (possibly from a number of sources) and updates the display as necessary.
- Stream - this process consists of any application which waits for a display data request. Upon receipt of a request, synchronous data is sent out for display purposes.
- Periodic data server - this process is similar to a stream, but generates asynchronous, periodic data at a frequency determined by the requesting process.
- Data Sampler - fork of the periodic data server.

A major part of the Operator Display Subsystem is the user interface which it presents. This is a completely application-defined (new) user interface which provides the following capabilities:

- Operation via mouse and menus.
- Alternative operation via keyboard and the command line.
- Flexible display layout.
- Graphics and color.
- Protected access to critical functions.

The PTPOCC user interface provides a completely new environment to the operator. It supports an interface which can be driven either by the mouse or the keyboard. Keyboard input is via TOL statements entered in a command line. There does not appear to be any escape to the native operating system or its command interpreter (UNIX shell). The interface alleviates this need by providing all functions necessary for the operator to use the system.

The PTPOCC user interface maintains a strictly defined display format. The display consists of several consistently placed areas for operator control, data display, and message display. A sample display appears in Figure 8-1 (the four functional areas are divided by bolder lines).

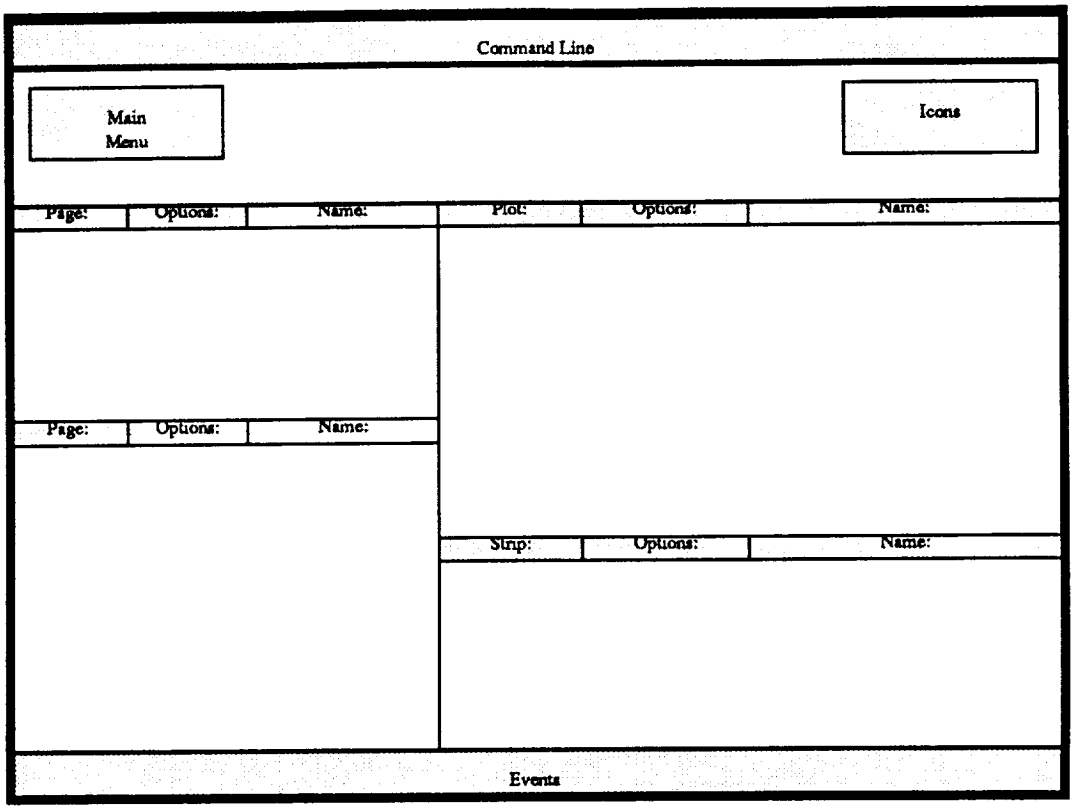


Figure 8-1 Sample PTPOCC Display

The PTPOCC user interface consists of four functional areas. From the top of the screen to the bottom, these include:

- Command line - allows entry of TOL command statements. System may be operated entirely from within this interface. This area consists of:
  - Input line - used for command entry. If a command is invalid, it is re-displayed for editing; if correct this line will be cleared.
  - Response line - a message will confirm command execution or will identify invalid part of command. Color will be used to include success or failure.
- Header area - area in which system information and main menu/icon structures appear:
  - Main menu - lists functions which affect the display as a whole including screens, options, telemetry, command, housekeeping, and terminate.
  - Icons - tile, window, or tool. When the operator moves one to the display area, it will expand to present the appropriate function.
- Display area - large area of the screen in which the following types of data displays may be placed:
  - Tiles - non-overlapping areas used to monitor telemetry data. Formats include alpha-numeric, strip-chart, meter, and x-y plot. Tiles include local menus for generic and tile-specific functions.
  - Windows - overlapping temporary areas used for information or prompts. As with tiles, windows include local menus.
- Events area - set of scrolling lines which present global system and local event messages.

Although the PTPOCC display format is fairly rigid, there remains adequate flexibility in the area used for data display. There are a number of functions which allow the operator to place and size the different tiles used for data display.

8.1.4.2.4 Event Subsystem

The Event Subsystem consists of a single process which is responsible for logging events and routing them to interested nodes. A remote node can set up a connection and specify a filter which allows event routing based on number and type. In such a case, the local events system will route applicable events to the interested node(s).

8.1.4.2.5 GMT Subsystem

The GMT subsystem consists of a single process which provides the time for local processes. It obtains the time from a local time code generator, a local system clock, or via a connection to a remote process.

8.1.4.2.6 Spacecraft Communications Subsystem

The Spacecraft Communications Subsystem is responsible for communicating with a NASCOM interface. It receives, records, and distributes telemetry frames. It also blocks, records, and sends command frames. This subsystem consists of three processes:

- Spacecraft Telemetry Communications Server - this process handles input of telemetry data. It reads telemetry data, stores it in a history file, and distributes it for real-time processing.
- Spacecraft Command Communications Server - this process handles output of commands. It buffers the commands and records them to a history file.
- NASCOM Channel Control - this process performs all NASCOM interface configuration and status monitoring functions.

8.1.4.2.7 Spacecraft Telemetry and Command Subsystem

The Spacecraft Telemetry and Command Subsystem performs telemetry decommutation, status checking, and command verification functions on telemetry frames. This subsystem also builds and outputs command frames. This subsystem consists of two processes:

- Spacecraft Telemetry Decommutation - telemetry decommutation is driven by a structure in a data base. This data base contains information which allows the telemetry data to be correctly decommutated. During this time, limits are checked and trend analysis is performed.
- Spacecraft Command Generation - this process constructs command frames, implements command retries, and initializes data for command verification.

8.1.4.2.8 Spacecraft Communications Simulation Subsystem

The Spacecraft Communications Simulation Subsystem performs the inverse function of the Spacecraft Communications Subsystem. This subsystem takes input from remote telemetry processes (normally the Spacecraft Telemetry and Command Simulation Subsystem) or from the telemetry history file and routes to the NASCOM interface. It also takes command input from the NASCOM interface and routes it to a history file or a remote command handling process (normally the Spacecraft Telemetry and Command Simulation Subsystem).

8.1.4.2.9 Spacecraft Telemetry and Command Simulation Subsystem

The Spacecraft Telemetry and Command Simulation subsystem performs the inverse function of the Spacecraft Telemetry and Command Subsystem. It commutates telemetry data for routing to other processes (normally the Spacecraft Communications Simulation Subsystem). It also decommutates commands taken from other processes (normally the Spacecraft Communications Simulation Subsystem).

8.1.4.3 PTPOCC Configurations

The network transparent manner in which the PTPOCC subsystems communicate allow processes to be located on different processors to meet the processing requirements of the system. For a system with minimal processing requirements, a real-time workstation processor could be used to execute all subsystems. The most efficient solution is to use a front-end, real-time system handling the communications and telemetry/command functions, one to many graphics workstations providing operator interaction, and another system providing simulation functions. The advantage to this configuration is that new workstations could be added to support more users (the most common required processing increase). This configuration would include the following processors which would in turn run the listed subsystems:

- Telemetry and Command Processing System:
  - Operator command.
  - Events.
  - GMT.
  - Spacecraft Telemetry and Command.
  - Spacecraft communications.
- Operator Interface System(s):
  - Operator command.
  - Operator display.
  - Events.
  - GMT.
- Telemetry and Command Simulation System:
  - Operator command.
  - Events.
  - GMT.
  - Spacecraft Telemetry and Command Simulation.
  - Spacecraft Communications Simulation.

8.1.4.4 PTPOCC Implementation Plans

The documentation reviewed for this survey did not indicate that the PTPOCC system was at a point in development to support the SMMOCC. Rather it was being implemented in phases on different types of hardware with various types of software. One of the problems is that some of the standards upon which the PTPOCC is based are not yet mature enough to easily use (at least they are not available as commercially supported products). One example is X Windows. For this reason, some of the software was forced to use non-standard software as an interim solution. Another problem is that the PTPOCC will be implemented on existing hardware (as opposed to specially

procured hardware). Therefore some of the interim prototypes do not represent the most effective configuration.

The list of prototype configurations which have been completed or are planned in the near future include the following:

- Operator Interface System (Sun implementation) - This processor is an operator workstation which relies on a separate processor to provide real-time data. It will use Sun's proprietary window system (SunView).
- Single Processor System - This is a real-time system based on a VME system running VxWorks. The operator interface will be provided an X terminal over the Ethernet.
- Telemetry and Command Processing and Simulation Systems - This system will expand upon the Single Processor System to include a complete implementation of the real-time communications, telemetry and command processing subsystems, and simulation subsystems.
- Operator Interface System (IBM RT implementation) - This processor is an operator workstation based on the IBM RT. It will use the X Windows for graphics support.
- SMMOCC Operation System - This system will be that used to support the operational requirements of the SMMOCC. It will require additional subsystems for existing interface communications and SMM-specific application requirements.

8.1.5 Applicability to the Concept Executive

The TPOCC concept provides a solid foundation for the design and implementation of a workstation executive which supports a small spacecraft control environment. Although smaller in scope, the TPOCC concept offers a number of design approaches which are applicable to the Concept Executive. The TPOCC concept defines a system which is based on industry standards, uses state-of-the-art workstations for processing and user interaction, and achieves configuration independence by distributing data over the network.

The TPOCC concept is completely based on industry standard hardware and software. The TPOCC concept will function on any hardware which meets the capability requirements and provides the required software. The TPOCC software is to be designed such that the primary functions are implemented in a standard manner and then supported with system-specific software. One way to accomplish this is to write standard "front-ends" to be used by the primary functions. These front-ends are then implemented using the proprietary functions of the host computer. Although this is a good approach when no standard exists, it is more efficient for the primary functions to directly use a standard (such as TCP/IP or X Windows). The TPOCC system will use standards when available and then provide front-ends for required functions which are not yet available as standards.

The majority of the software which makes up the Concept Executive can and will be based on standards. However there are some requirements which may dictate use of proprietary features for the sake of performance (data display) or lack of an available standard (real-time functions). One solution to such problems is to develop standard front-end functions to be used by the executive software. These front-ends are then implemented via the proprietary functions provided by the host computer. This approach can be extended in that front-ends can be make to look and behave like proposed standards (such as POSIX real-time extensions). In this way, software developed now will not have to go through a major porting effort when full standards are available later.

Perhaps the most significant concept in the TPOCC is the ability to transparently distribute real-time data in a single processor or between multiple processors over the network. This concept allows a great deal of flexibility in terms of hardware configurations. The TPOCC concept uses inexpensive and isolated real-time processors for the actual real-time data acquisition. This approach isolates the true real-time requirements on the appropriate processor. In this way, common general-purpose workstations may be used for less real-time functions such as data manipulation and display. Although this approach increases the complexity of the software, it allows more cost-effective and efficient configurations to be developed. Note that a valid argument is in the case where absolute real-time response is required for the entire data throughput path (acquisition, manipulation, and display). In such a case, it may be necessary to use a general-purpose, tightly-coupled real-time processor for all functions.

One of the goals of the Concept Executive is to support many different configurations of user workstations. An example is a server processor which is directly connected to the global networks providing all data. This processor would then provide data to other processors (connected via a subnet) which actually do the data processing and display. To accomplish this goal, it must be possible to efficiently distribute large amounts of data across the local network. The TPOCC concept offers one means of achieving this goal. However, it must be determined if this approach is efficient enough for large amounts of data.

The TPOCC concept provides a complete user interface. This user interface is keyboard and mouse based and should alleviate the requirement to ever access the native operating system command line interpreter. This user interface includes its own command language (described below) and a well-defined window/mouse interface. The user interface attempts to provide all functions which a user would ever need, therefore eliminating the need to access the operating system directly. One disadvantage of the interface is that the display format is rigidly defined. Various system and message windows are placed in standard locations and cannot be removed or repositioned.

The TPOCC system provides a complete command language (TPOCC Operations Language) from which all functions in the system may be controlled. This language is used for both interactive and programmatic usage and is the primary programming interface for the user. The combination of this language and the display definition tools provides a programming environment which should satisfy the operational requirements of users. This language is intended to support all basic operational functions. It is expanded to support operations which are specific to the current project (satellite). Use of this language precludes the requirement of using either the native command line or developing actual host programs (C, FORTRAN, etc.). This approach insures portability of application software and provides the basis for certifying that an application will not perform an illegal action during operational use.

Providing a complete user command language is not considered part of the Concept Executive. Developing a new language would be useful for configuration management and user support functions, but would require a great deal of development effort. It is also not reasonable to burden the Concept Executive with this function, as it may not be necessary for all environments. If such an interface is required, it should be developed as a layer above the Concept Executive.

8.2 GCS System

Kennedy Space Center (KSC) currently utilizes a complex distributed processing system called the Launch Processing System (LPS) to checkout, control, and monitor space equipment being readied

for launch. Although this system was state-of-the-art when developed, it now has many limitations, including:

- Limited hardware reconfigurability.
- Obsolete hardware.
- A large amount of custom hardware.
- Limited processor memory.
- A difficult test reconfiguration procedure.

Due to the problems listed above and to the increased launch load expected in the near future due to space station and increased shuttle activity, it becomes obvious that a new launch checkout system is required. The result is a set of specifications for the Generic Checkout System (GCS). The GCS will use the latest in hardware and software technology, will retain the best features of the existing LPS, eliminate the existing limitations, and serve as the foundation upon which unique test, checkout, and monitor systems are developed.

The Generic Checkout System is intended to serve as the foundation for the Test, Control, and Monitor Subsystem (TCMS), which will be used to support Space Station testing. The GCS will also be used as the basis for the replacement Checkout, Control, and Monitor Subsystem (CCMS) responsible for shuttle support (the new system is called the CCMS II).

The Generic Checkout System (or a subset of it) is also referred to as "core" or the "generic" system. All terms are interchangeable and refer to the fundamental system concept. For the remainder of this document, the term "generic system" will be used.

8.2.1 Contact Point

Engineering Development Directorate

Kennedy Space Center

8.2.2 Review Process

The review process for the generic system involved examination of the following documents:

- Core Electronics System Specification.
- Ground Data Management System (GDMS) Facility and Equipment Design Plan.

The specification document provided a good overview of the generic system concept. This document was primarily a requirements and specifications document and did not provide detailed design. However, it still provided a good foundation for review.

8.2.3 Generic System Requirements

The high-level requirements for the generic system are presented in the following subsections. These requirements only represent the generic system; they do not include specific requirements for either the TCMS or CCMS II systems.

8.2.3.1 Programmatic Requirements

Programmatic requirements define the basic design direction of the generic system. The programmatic requirements are as follows:

- The system design shall support a 30-year life cycle in a cost-effective manner.

- The system shall primarily consist of COTS hardware and software. It will use widely supported commercial standards.
- System flexibility shall be a primary goal. Vendor-independent implementations shall be used whenever practical.
- The system shall be comprised of modular subsystems. Widely supported interface techniques shall be used. All subsystem hardware and software modules shall be upgradable with minimal or no changes to other subsystems.
- The system shall be designed to allow incorporation of, and seamless transition to new hardware and software technologies.
- All subsystem modules shall comply with industry standard hardware and software interfaces and protocols. Compliance shall include, but not be limited to:
  - Standard programming languages.
  - Utilities.
  - Operating system interfaces.
  - Communication protocols.
- The system shall have the capability to be logically and physically partitioned into independent subsets. The capability of a subset may vary from a single element test to a large integrated test.
- The system shall provide the capability for fast and cost-effective maintenance of the system life cycle.
- The system shall provide automated tools for the rapid and efficient detection and isolation of system faults and failures.

8.2.3.2 Functional Requirements

Functional requirements define the capabilities provided by the system. The functional requirements are as follows:

- The system operating system shall allow users to create applications which are independent of any particular subsystem processor.
- The system shall provide the capability of defining, managing, and allocating system resources to support a given test configuration.
- The system shall provide the capability to build, load, and modify test configurations.
- The system shall provide the capability to define, build, maintain, and utilize data bases of test equipment characterization and definition data.
- The system shall provide the capability to develop, verify, and implement user application software.
- The system shall provide the capability to access data bases on different subsystems within a test configuration. Access will be via a standard query language.
- The system shall provide the capability to record and archive test and transaction data.
- The system shall provide data analysis and data reduction capabilities to assist a user in assessing test element and/or system performance.

- The system shall provide the capability to perform an end-to-end operational readiness test to verify a test configuration.
- The system hardware will operate within the available power capability of the facility.
- The system shall provide an equipment health and status monitoring capability.
- The system shall provide the capability for manual or automatic switch-over to redundant systems.
- When subsystems connected to a network are powered up or down, this shall not disrupt communications over the network.
- The operational status of backup subsystems shall be monitored to ensure availability.
- The system shall provide an independent safing capability for the emergency control of critical and/or hazardous systems.
- The system shall provide security to protect the system and test elements from access by unauthorized users.
- No command shall be executed without validation of the command source and context.
- No recoverable error shall halt system-execution or data acquisition processing.
- Automatic correction of data and memory errors shall not be hidden from the system. Notification of any such corrective action shall be made available.
- The system shall function within what is considered an electronically hostile environment.
- The system shall provide the capability to direct test and graphic files and screen images to local or remote storage and/or output peripheral devices.
- The system shall provide, through a Data Security System, the capability to monitor for unauthorized attempts to gain access.

8.2.3.2.1 Interface Requirements

Interface requirements define the manner in which the system interfaces with other external systems. The interface requirements are as follows:

- The system shall provide the capability to interface with external processing and monitoring systems.
- The system shall provide transient protection on all power supplies and input/output hardware interfaces.
- Commercially purchased equipment shall have protection provided on all processed data input/outputs to the extent specified in commercial specifications.
- Purchased interfaces shall have protection up to the extent of the manufacturer's standard practice.

8.2.3.2.2 Performance Requirements

Performance requirements define the ability of the system to acquire and process test data. The performance requirements are as follows:

- The system throughput requirement is 50,000 significant changes of measurements per second.

- The system shall be capable of responding to at least 100 measurement exceptions per second on any single front-end data link.
- The system shall be capable of responding to at least 400 measurement exceptions per second on all front-end data links.
- The system shall be capable of supporting a total of at least 700 subsystem-to-subsystem transactions per second.
- The system shall provide a mechanism that will prohibit updates of multi-word data while that data is being accessed. This mechanism shall support a total of up to 1000 accesses of this type per second.
- The system shall support the capability for any application program to access the current value in shared memory of any measurement in less than 1 millisecond.
- The system shall support the capability for any application program to continuously access a single measurement at least 200 times per second.
- The system shall allow update of display data on a user-generated skeleton at least once per second (not including display skeletons which are not visible).
- The system response time for a reactive sequence shall be less than 50 milliseconds.
- The system shall be able to increase overall performance to meet new requirements by expanding subsystem performance and capacities to the maximums required within the subsystem specifications. Increase is via better performance, capacities, and new technologies.

8.2.3.2.3 Other Requirements

This section addresses miscellaneous system requirements. These requirements are as follows:

- The system shall provide an Expert System development environment. This environment will include all software tools, utilities, and aids to develop and validate artificial intelligence and expert system user-level and system-level applications.
- The system shall provide access to all data within a test configuration by all users, subject to a user's permission level.
- The system shall provide the capability for users and applications to asynchronously route formatted and unformatted commands, data, and information to other users and applications on the system.
- The system shall provide an Operations Support function controlling test configuration equipment power-up, system and operational software load, and test configuration activation and deactivation.
- Subsystem components shall be modularly replaceable and interchangeable and shall be configured to allow easy repair, replacement, or upgrade.
- On-line maintenance and diagnostic functions will be accessible from any workstation in a test configuration.
- A system-availability model, such as the Automated Reliability Assessment Model developed by the Langley Research Center, shall be used to verify that subsystem MTBF and MTBR data supports system availability requirements.

- User-room workstation housings shall be comprised of modular, movable, and reconfigurable components.
- Custom designs, boards in the Data Acquisition Module and the Remote Interface Module supporting items under test shall be designed such that the final output can be verified through an independent channel on the board.
- Failure of a subsystem on the network shall not interfere with the normal operations or communications of the rest of the subsystems on the network.
- Each subsystem shall maintain a status log (success or failure) of transactions it performs on the network.
- A loss of redundancy shall be treated as a system error, and the system shall generate the appropriate messages to the operators to provide this information.

8.2.4 System Description

The generic system provides a general-purpose, cost effective, flexible, real-time, test and checkout system capability. This system defines the necessary hardware, software, tools, environments, and support to develop and execute applications to perform integration and testing for a variety of space systems. The generic system is not an implementation, but a concept for the functions and capabilities required for specific test and checkout systems (such as for shuttle and Space Station).

The TCMS and CCMS II systems are the first applications of the generic concept. All generic concepts and requirements are fully represented in both the TCMS and CCMS II systems. Each system also includes application-specific extensions required to support the particular environment. A discussion of these extensions is beyond the scope of this document.

The generic system consists of a core system with front-end and back-end interface extensions. The central portion shall consist of a distributed network of real-time processors. All processors will utilize standard network interfaces to allow additional processors to be added and logical "subsets" defined for the requirements of a particular test. A subset is a logical association of front-end, core, and back-end processors.

The front-end extension shall consists of a number of independent clusters of Data Acquisition Subsystems (DAS). Each cluster will contain some number of Data Acquisition Modules (DAM) connected to a single Data Acquisition Processor (DAP). The DAM's directly access the hardware to be tested. They perform all hardware-dependent processing and route the data to the DAP. The DAP accumulates the data and concentrates it into a standard format for use by other processing subsystems.

The core system will consist of a number of Application Processor Subsystems (APS) upon which the majority of the data processing applications are executed.

The back-end extension will consist of clusters of workstations. These Display Processors (DPS) provide the user interface to the core system. These workstations will provide a multi-window graphic interface which incorporates the latest in display technology.

This architecture will consist of five major subsystems which communicate via two network subsystems. The subsystems include the following:

- Data Acquisition Subsystem (DAS).
- Application Processor Subsystem (APS).

- Archive and Retrieval Subsystem (ARS).
- Data Base Subsystem (DBS).
- Display Processor Subsystem (DPS).
- Global Network Subsystem (GNS).
- Display network Subsystem (DNS).

Figure 8-2 illustrates the configuration of the different subsystems. The dotted line outlines one possible test configuration subset (subsets are explained in the following discussion).

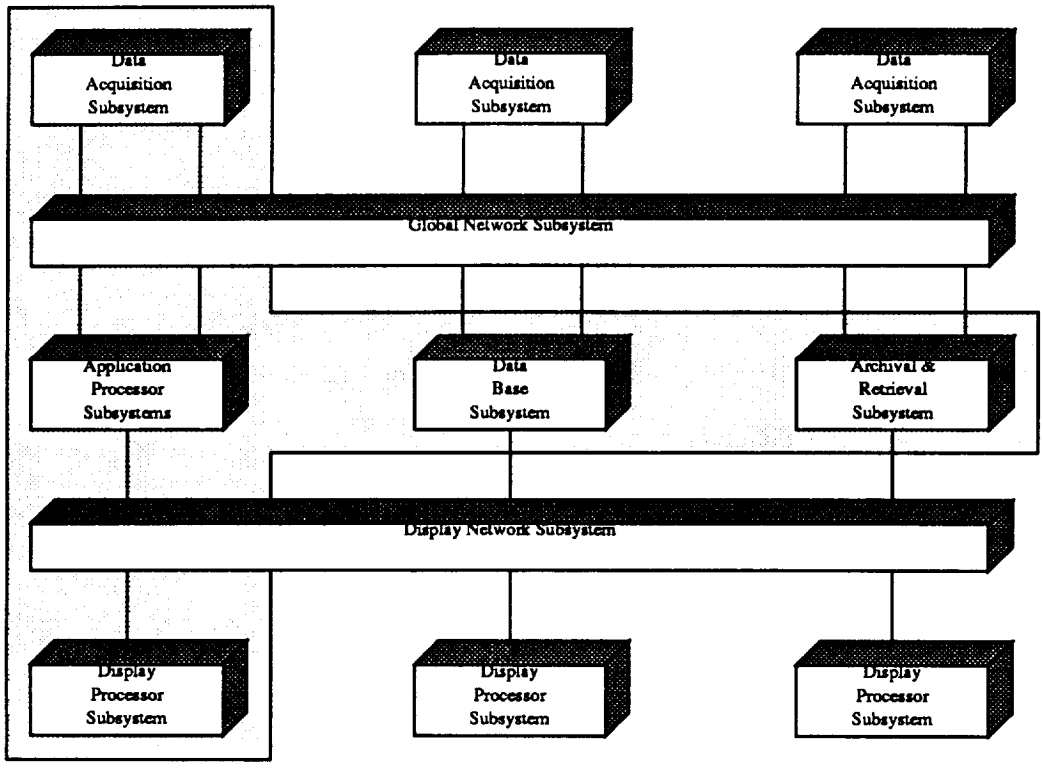


Figure 8-2 Generic Checkout System

In addition to the major subsystems described above, there are additional subsystems which provide specialized functions and capabilities:

- Digital Record and Retrieval Subsystem (DRRS).
- Configuration, Calibration, and Test Sets (CCATS).
- Remote Interface Modules (RIM).
- Software Production Facility (SPF).

The Global Network Subsystem will allow the system to be partitioned into independent logical subsets for testing purposes. A subset defines a way of logically associating (via the network) the required DAS, APS, DPS, DBS, and ARS subsystems. The ability to dynamically reconfigure the system supports a variety of test requirements with a single integrated system. Note that a subset will consist of the two network subsystems and at least one of each of the remaining five major subsystems (DAS, APS, DBS, ARS, and DPS).

The generic system data flow involves taking raw data from the test element into the Data Acquisition Subsystem, converting it to a standard format and distributing it on the Global Network Subsystem, using the Application Processor Subsystem to process the data and generate test commands for return to the Data Acquisition Subsystem. During this time, the Display Processing Subsystem is used by the test operator to monitor and control the test. In addition, the Archive and Retrieval Subsystem and the Data Base Subsystem are used to save and retrieve test data.

Figure 8-3 illustrates the generic system data flow between each of the major subsystems.

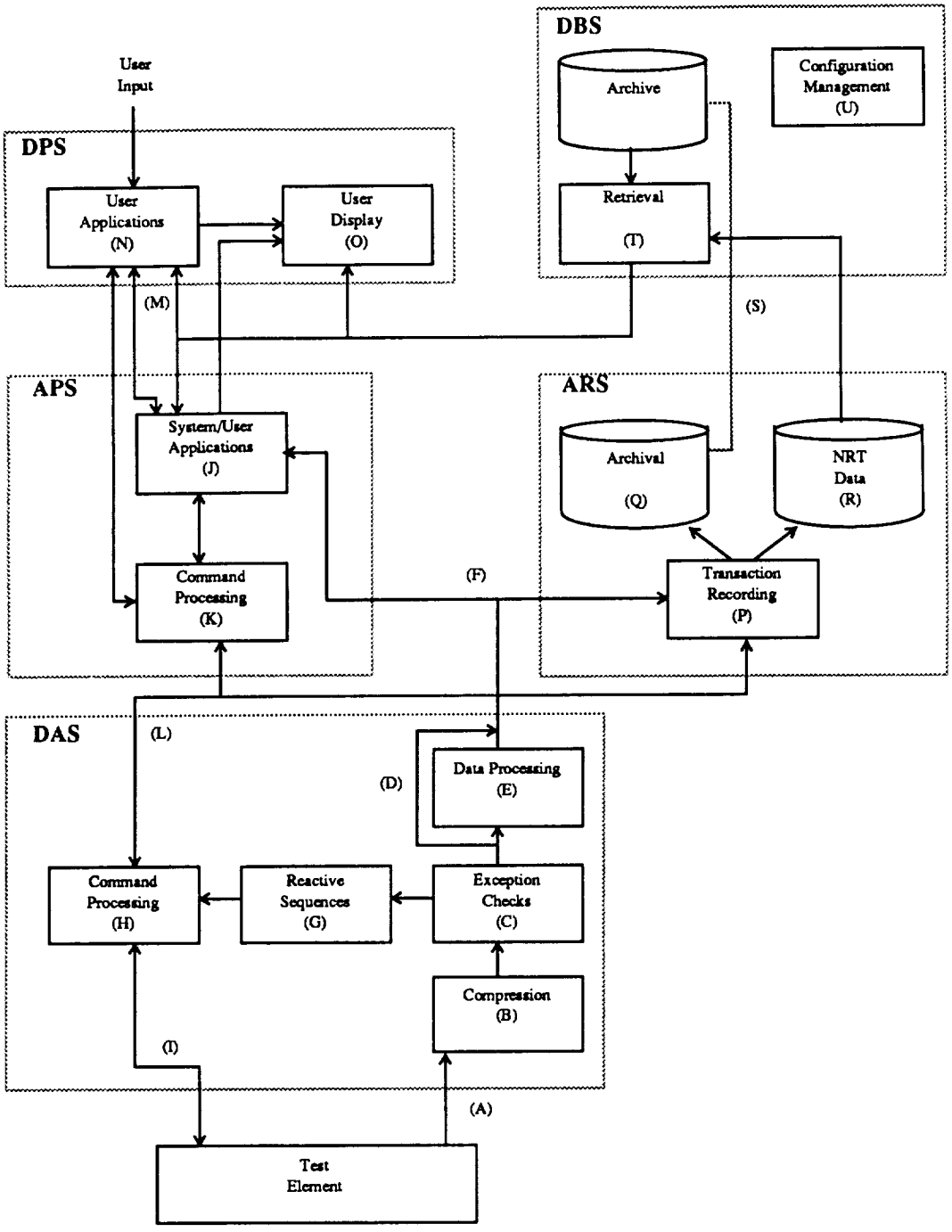


Figure 8-3 Generic Checkout System Data Flow

The following subsections provide a more detailed description of the processes within the subsystems and the flow of data between them.

8.2.4.1 Data Acquisition Subsystem (DAS)

The Data Acquisition Subsystem (DAS) processes all responses from and commands (H) to the test element. The DAS takes standard commands and converts them into the format required by the

test element. Similarly, the DAS converts test element-specific data to a standard format for distribution to the Application Processor Subsystem.

The DAS retrieves test element-specific data (A) from the test element. This data is then compressed (B) and examined to determine if any exceptional values were retrieved (C). If an exception occurs, a reactive sequence may be initiated (G). The reactive sequence will initiate a command to enact the appropriate action on the test element.

The compressed data is then processed (E) and released in this form. Approximately 20 percent of the data is also released in raw form (as opposed to processed). All data is distributed in a standard format.

The Data Acquisition Subsystem consists of a Data Acquisition Module (DAM) and a Data Acquisition Processor (DAP). The DAS may consist of 1 to 16 DAM's and 1 or 2 DAP's. The purpose of the DAM is to implement both standard tests and those unique to the test element. The DAM isolates the specifics of the test element from the rest of the system (and the rest of the DAS). The basic functions of the DAM are:

- Data compression.
- Exception checking.
- Linearization.
- Trend checking.
- Data checking.
- Engineering units conversion.

The DAP provides the functions required to concentrate and communicate the data to the remainder of the processing system. The basic functions of the DAP are:

- Data concentration.
- Reactive sequences.
- Command validation and authentication.
- Table management.
- Message routing.
- Health checking.

8.2.4.2 Application Processor Subsystem (APS)

The Application Processor Subsystem (APS) provides the primary execution environment for test applications, which include system and user programs. Applications in the APS will monitor data from the Data Acquisition Subsystem (F) for necessary test information. This information may be sent to a Display Processor Subsystem for display (O), relayed to other applications in the APS and DPS (M and N) for further processing, or used to generate a command. All generated commands are routed through the command processor (K). This command processor validates the command and performs any initialization required for proper command execution.

The APS will consist of 1 or more Application Processors connected via the network. The APS interfaces to the Data Acquisition Subsystem via the Global Network Subsystem and the Display Processing Subsystem via the Display Network Subsystem. The APS will execute system func-

tions such as control of user access, managing system resources, and performing system diagnostics.

8.2.4.3 Display Processor Subsystem (DPS)

The Display Processor Subsystem provides the system operator interface. The user display process (O) presents data on the state of the entire test system. Displays may be driven by applications in the DPS (N), Application Processor Subsystem (J), or from data in the Data Base Subsystem (T). The DPS will provide mouse input processing (with keyboard alternative), window management, and other user support functions.

The DPS is the back-end extension to the processor core. The DPS will present a workstation-based user interface which is efficient and consistent across all applications. This will simplify the operators interaction and minimize the required training. The DPS will consist of one or more high-performance, color, engineering-level workstations, as required by the number of users.

The DPS will provide all functions necessary for efficient man-machine interaction. This includes:

- Local storage of display skeletons and programs.
- On-line build and modify of displays.
- Limited local processing of user programs.
- Remote processing capability.
- Multi-window environment.
- Simultaneous virtual machine sessions.
- Real-time display of measurement data.

In addition to providing user-level windows, the DPS will provide a number of system-generated windows. This includes windows for:

- System messages.
- Exception messages.
- Data display.
- System status.
- Test configuration and control.

8.2.4.4 Archive and Retrieval Subsystem (ARS)

The Archive and Retrieval Subsystem records the commands and data exchanged between the Data Acquisition Subsystem and the Application Processor Subsystem. The recording process (P) monitors the commands (L) and data (F) traffic and records the information on:

- Two permanent archive records.
- A temporary near-real-time (NRT) record.

The permanent archive record will consist of a permanent archive media (such as optical disks) and a removable media which is transferred to the Database System (DBS) when appropriate. The ARS will interface with the DBS to provide access to temporary NRT data or any archived data. This data is transferred to the DBS from which it may be retrieved by a user or application.

8.2.4.5 Database Subsystem

The Data Base Subsystem (DBS) provides retrieval and playback ability for examining stored test information. The retrieval process (T) takes test data from archives in the DBS, from NRT temporary media in the Archive and Retrieval Subsystem, or from a permanent archive in the ARS. This data is formatted and distributed to the requesting user (O) or application in the Display Processing Subsystem (N) or the Application Processor Subsystem (J).

The DBS provides a configuration management process (U) which affects the operation of all subsystems. The Software Production Facility (SPF) shall contain the "super-set" of all application libraries, data bases, and tables. The DBS is used to store the subset of this information which pertains to the current test configuration. Although the SPF is the focal point for configuration management, the DBS provides configuration management during test operations.

8.2.4.6 Global Network Subsystem (GNS)

The purpose of the two network subsystems is to allow data, commands, and display information to be exchanged between processor subsystems. The networks also allow independent subsets to be defined to support multiple test activity within one integrated system. The Global Network Subsystem provides the following capabilities:

- Routing of information (mostly test data/commands) between DAS's, APS's, ARS's, DBS's, and DRRS's. This routing is performed on global communication and data buses.
- Connections for remote maintenance of each of the subsystems listed above plus all network devices.
- Ability to isolate groups of subsystems into subsets.
- Redundant path and switch-over abilities.

8.2.4.7 Display Network Subsystem (DPS)

The Display Network Subsystem will consist of a number of local display buses, each of which is in turn connected to the global display bus via a bridge. The DPS subsystem will provide the following capabilities:

- The ability to route data (mostly workstation display, command, or retrieval data) between DPS's, APS's, ARS's, DBS's, DRRS's, user-provided equipment, the Software Production Facility, and external systems.
- The ability to isolate DPS's and APS's into local display buses with bridged interfaces to the global display bus.
- The ability to control all local display bus to global display bus interfaces.
- The ability to connect DP's, DBS's, ARS's, DRRS's, user-provided equipment, and external system interfaces to the global display bus.
- The ability to access any subsystem on the DNS from a user workstation (assuming user has the appropriate permissions).

8.2.4.8 Digital Record and Retrieval Subsystem (DRRS)

The Digital Record and Retrieval Subsystem includes a process (V) which records the command (I) and data (A) exchanged between the test element and the Data Acquisition Subsystem. This

information may be retrieved and used by applications executing on the Display Processing Subsystem and the Application Processor Subsystem.

8.2.4.9 Configuration, Calibration, and Test Set Subsystem (CCATS)

The Configuration, Calibration, and Test Set Subsystem provides the ability to monitor any system data bus (Global Network Subsystem, Display Network Subsystem). This is accomplished via use of commercial and custom network analyzers, common generic subsystems, and special test equipment.

8.2.4.10 Remote Interface Module (RIM)

The Remote Interface Module provides the interface between the Data Acquisition Module (DAM) of the DAS and the Ground or Flight support equipment.

8.2.4.11 Software Production Facility (SPF)

The Software Production Facility provides an environment for development, integration, testing, configuration management, and maintenance of user and system software. This includes all tools required for effective software development and maintenance. The functions provided by the SPF include:

- Software management support (word processing, scheduling, and project management).
- Software production tools:
  - User application software compilers.
  - System application software compilers.
  - Editors, debuggers, linkers, configuration tools.
- Database creation and maintenance:
  - Test article data base.
  - Measurement and command data base.
  - Hardware and software test configuration management.
- Test management tools.
- Simulation support:
  - Model development.
  - Simulation software development and verification.
  - Operator training.
  - Test support.

The SPF will provide a central host and a network of workstations to provide the development environment. This environment will be logically isolated from the operational test system. The SPF will be connected to the generic system via the Global Network Subsystem.

8.2.4.12 Software Description

The software for the generic system will consist of system and support software. System software is that which is common to all subsystems. Support software is that used on specific subsystems. The remainder of this section will discuss the system software of the generic system as it most clearly defines "executive" functions.

The generic system will function in one of several operational modes. These include the following:

- Test development - provides the functions required to support user software development.
- Configuration - provides the functions necessary to configure a subsystem for operational use.
- Operations - provides functions required for the generic system to interface with the test element.
- Maintenance - provides functions necessary to support system diagnostics and maintenance.

The system software is divided into the following functions (the following subsections will discuss these functions in more detail):

- Operations support.
- System configuration.
- System maintenance.
- Operating system and utilities.
- Network interface.
- System load and initialization.
- System integrity.
- Diagnostic aids.
- System security.
- Test application execution.
- System messages.

8.2.4.12.1 Operations Support

The Operations Support function will execute in an Application Processor Subsystem of each test subset. This function is the central point from which the test subset may be initialized, monitored, and operated. This function provides the following capabilities:

- Coordination and control of power-up, initialization, and operational software downloads.
- Failure detection and corrective actions including subset reconfiguration (correction is via interface to the System Configuration function).
- Hierarchal status displays with increased levels of detail.
- Configuration displays for overviews or details of the subset configuration.
- Recording of configuration and health data for later use in subset configuration verification.
- Status reporting of any command, measurement, subsystem, and subset configuration.
- Monitoring for configuration changes.
- Data between control rooms will be interchangeable via a standard data base.
- Access to the Problem Reporting and Corrective Action System.

- Remote configuration management only from a specific Application Processor.
- All subsystems and components on the network will provide health and status information.

8.2.4.12.2 System Configuration

The System Configuration function will execute in each facility supported by the generic system. This function will execute on a designated Application Processor. This function maintains the configuration and health/status of all networks and subsystems. The System Configuration function will interface with the Operations Support function in each subset to detect failures and initiate the required corrective actions.

8.2.4.12.3 System Maintenance

The System Maintenance function will execute in each facility supported by the generic system. This function will execute on a designated Application Processor (normally the same as used for the System Configuration function). This function provides on-line maintenance capabilities including:

- Determining the status, availability, and configuration of system hardware.
- Monitoring, analyzing, testing, and troubleshooting hardware and network failures.
- Executing diagnostics in all subsystems (except Display Processors).
- Performing on and off-line diagnostics in all subsystems (except Display Processors).
- Diagnosing faults at the subsystem level and then isolating the fault to the particular module.
- Performing subsystem data dumps.

8.2.4.12.4 Operating System and Utilities

The generic system's operating system will allow users to develop applications which are independent of any subsystem processor. The Operating System and Utilities will be a common development environment which is consistent across all subsystem processors. This operating system will initially conform to POSIX 1003.1, but must later adhere to the complete POSIX description. Note that IEEE standard 1003.1 only specifies the programmatic operating system interface. This is being expanded to include command line access, networking, real-time extensions, etc.

The generic system's operating system must provide real-time extensions for use on the DAP, AP, DRRS, and ARS subsystems. The real-time mechanisms to be provided include:

- Scheduler supports fixed priorities which do not age (as normal UNIX).
- Process dispatch latency of less than 10 milliseconds (amount of time it takes a blocked process to respond to an event).
- Time of day must be available in resolution of milliseconds.
- Timers supporting resolution in the microsecond range.
- Asynchronous input/output.
- Ability to lock a process (or portion) in core memory.

8.2.4.12.5 Network Interface

The Network Interface function is based upon the Ethernet standard. Transmission Control Protocol/Internet Protocol (TCP/IP) will be used for data transport. In the future, all network access will adhere to the International Standards Organization's (ISO) implementation of the Open System Interconnect (OSI) 7 layer model when this standard is defined and available.

8.2.4.12.6 System Load and Initialization

The System Load and Initialization function provides the ability to initialize the system to support the test configuration. This function provides the capability for the following forms of system initialization:

- Network configuration - allows the network to be logically partitioned into test configurations.
- System load and initialization - allows a subsystem to be loaded with all required system software.
- Subsystem Initialization - allows a subsystem to be initialized and configured to allow operational use.
- Test application distribution - allows a subsystem to be loaded with the test applications required to support a particular test configuration.

8.2.4.12.7 System Integrity

The System Integrity function monitors the health of the system and subsystems to insure their reliability during operational use. This function provides the capability for the following:

- Subsystem health and status - determines the health and status of a subsystem and all attached peripherals.
- System health monitor - part of the Operations Support function. Retrieves the information from each subsystem health and status functions and provides hierarchal displays.
- Health data analysis - the health data will be archived on the Archive and Retrieval Subsystem. This data will be used for long-time health reports, failure trends, health over a time period, failure and re-try attempt counts, and system redundancy status.
- Operation readiness test - verifies that a test configuration is prepared to support operational use.
- System redundancy management - ability to control the switch-over to a redundant system during a failure.

8.2.4.12.8 Diagnostic Aids

The Diagnostic Aids function provides the ability for a subsystem to determine its own health and status. This includes the ability to perform a self test, provide on-line fault reporting, and system debugging tools.

8.2.4.12.9 System Security

The System Security function provides a complete hierarchy of protections and permissions which range from individual users to system-wide. This function will provide the following forms of security:

- User account security - all users will have individual user accounts. These accounts will be strictly controlled to prevent access to other user's or system data and functions.
- Use account administration - user accounts will be administered via a menu-driven application which is only accessible by specific users.
- Super user capabilities - super user capabilities will be restricted to system administrators.
- Process execution security - execution of programs during operational mode is restricted. Users will only be allowed to execute programs for which they are allowed.
- User privilege restrictions - privileges will be restricted on a per user or group basis.
- Command issuance security - restrictions will be placed on issuance of commands to the test element.
- Proprietary data protection - data considered private by a user will not be available for use by other users.

8.2.4.12.10 Test Application Execution

The Test Application Execution function provides the services required by an application to perform a test. This function will provide the appropriate libraries and services for use on all processors on which user applications may reside, including the Application Processor, Display Processor, and Data Acquisition Processor. This function will provide the following services:

- Real-time checkout support library - provides bindings for all languages supported by the development environment. The following general capabilities are provided:
  - Access and control of user displays and windows.
  - Control of events which initiate interrupts.
  - Read current value of time/data, timers, or measurement data.
  - Read/modify measurement processing parameters.

8.2.4.12.11 System Messages

The System Messages function will provide users and applications with messages corresponding to various types of system events. All messages will be presented in a meaningful, English-like syntax. The different types of messages will include:

- Fatal system errors.
- Fatal subsystem errors.
- Warning messages.
- Informational messages.

8.2.5 Applicability to the Concept Executive

The Generic Checkout System is a concept which includes many functions applicable to the Concept Executive. While the generic system is much larger in scope and does not directly define an executive level of software, it still provides concepts which may be applied. From a requirements point of view, the generic and Concept Executive systems are similar. Note that documents reviewed for the generic system were high-level specifications and did not discuss the actual software design. Therefore it was not possible at this time to evaluate or apply design and implementation concepts.

The generic system is heavily based on standard hardware and software. This includes all software to be based on the full implementation of POSIX when that standard is available. All network communications are also to be based on current standards (TCP/IP) and then migrate to ISO's implementation of the OSI model when that standard is available. This allows system and application software to easily port to replacement or additional processors. Use of standards allows the environment to grow as necessary to support additional test requirements. This also allows more efficient processors to be added as they become available.

The generic system includes a well-defined health and status function which monitors all networks and subsystems. This function also automatically and dynamically reconfigures the system to recover from faults. The Concept Executive should include an integrated function of this type which gathers the data needed to evaluate the status of all workstations, processes, and local networks. The executive should also automatically determine its local status and be capable of responding to failures.

A significant part of the functionality provided by the generic system is directed towards dealing with different types of spacecraft hardware which must be tested. This makes the interface to the test hardware a significant part of the system, as it must be capable of testing many different systems. Another unique function of the generic system is the ability to partition the system into subsets for testing purposes. Neither function is a major concern for the Concept Executive.

The generic system is fully distributed in that different major subsystems exist on separate processors (or clusters of processors). The primary functions of data acquisition, test application execution, and user interaction are present on separate processors. This separation allows expansion by adding processors and allows processors to be used for the purpose they are best suited. This approach could be used in the Concept Executive to distribute processing based on hardware capabilities.

8.3 MAE System

The Multi-satellite Support Operations Control Center (MSOCC) Application Executive (MAE) is the system currently in use at Goddard Space Center. Goddard Space Flight Center is responsible for support of a large number of satellite spacecraft. The MAE software system executes on a Concurrent 3280 applications processor (AP). The MAE system provides a generic set of services required by different satellite spacecraft. The basic functionality provided by MAE is then expanded to support the mission-specific requirements of individual satellites.

8.3.1 Contact Point

Code 510 and 511

Goddard Space Flight Center

8.3.2 Review Process

The review process for the MAE system involved examination of the following document:

- Multi-Satellite Operations Control Center (MSOCC) Applications Executive (MAE) Programmer's Guide.

The document reviewed is intended for an application programmer who is planning to use the executive to develop mission-specific software. As such it provides little overview material and is geared towards interfacing to the functions provided by the executive. This includes a detailed description of available functions, parameters, and global (FORTRAN common) data.

8.3.3 System Description

When developing a control center for a satellite, mission programmers will use MAE as the foundation for the system software. MAE provides a generic, primitive set of functions which support the basic requirements of a mission. Mission programmers will interface with MAE and add the required mission-specific software. MAE is a true executive as it provides the core set of functions required by applications programmers. This is necessary in such an environment as a control center environment must be developed for each satellite. Use of a common executive such as MAE reduces the amount of custom code which must be developed for each such mission. Figure 8-4 illustrates the relationship of MAE to other system hardware and software.

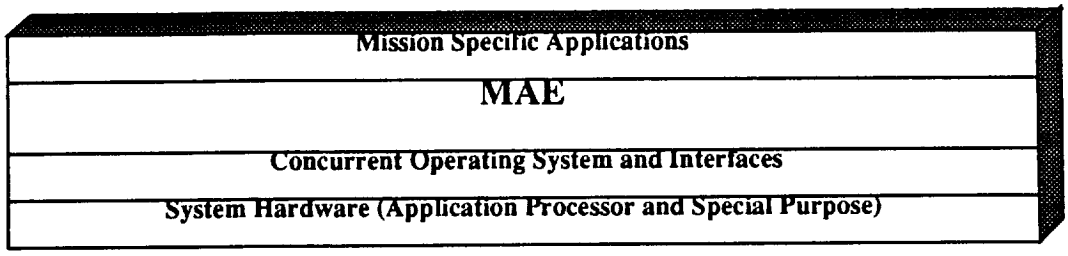


Figure 8-4 Location of MAE in System Hierarchy

The dividing line between the functionality provided by MAE and that left to mission-specific software is somewhat arbitrary. Nevertheless, the level of functionality provided by MAE is a good example of what an executive should provide to its application programmers.

8.3.3.1 Functional Overview

A complete mission control center system consists of MAE and mission supplied subsystems. The subsystems provided by MAE include the following:

- External interfaces.
- Display.
- System Test and Operations Language (STOL).
- Network Control Center (NCC).
- History.
- Database.
- Library.

The Display Subsystem includes only a subset of the required functionality. The subsystems which must be provided by the mission programmer include the following:

- Telemetry.
- Command.
- Off-line.

The mission developer must add software for the above listed subsections. For the telemetry and command subsystems, functions exist for allocating buffers and communicating with the NASCOM interface which communicates with the spacecraft (described in External Interfaces). The majority of the software described is used in an on-line manner. Mission developers must provide a separate off-line system as required.

The following section briefly describes the functionality provided by the MAE-supplied subsystems.

8.3.3.2 External Interfaces Subsystem

The External Interfaces Subsystem provides access to the external systems associated with a mission. This includes interfaces for the following:

- Idle System.
- NASCOM.
- Telemetry and Command (TAC).
- Terminal Interface.
- NCC.
- Device Configuration.

Each of these interfaces will be described in the following sections. Most of the interfaces (and other functions) require the use of buffers. MAE provides a number of buffer pools which may be used for this function. MAE provides the following buffer pools:

- (1) NASCOM input.
- (1) NASCOM output.
- (12) Mission Operations Division Local Area Network (MODLAN) pools (used by Idle System and on-line software).

Buffer pools consist of a varying number of buffers ranging from 5 to 30. The sizes of the buffers in a pool ranges from 128 bytes to 4094 bytes.

MAE provides a number of functions for getting, returning, and linking buffer pools.

8.3.3.2.1 Idle System Interface

The Idle System is a mission independent task which is separate from the on-line mission support system. The Idle System consists of a set of tasks when the AP is booted and not executing the tasks which are required for mission support (the system is "idle" as it is not controlling a mission).

The Idle System interfaces to the Mission Operations Division Local Area Network (MODLAN). This system establishes sessions and allows file transfer to and from hosts on the MODLAN. The Idle System Interface allows direct control over Idle System functions (sessions, file transfer) via STOL directives.

Communication with the hosts on the MODLAN is handled by the Network Executive (NETEX) software system. This package is responsible for routing and session initialization. From an application programmers perspective, the following functions are provided:

- Offer - offer connection services to other hosts.
- Connect - establish a session with another host.
- Read - read data from a host.
- Write - write data to a host.
- Close - normal session termination.
- Disconnect - abnormal session termination.

8.3.3.2.2 NASCOM Interface

The NASCOM Interface is used by all AP applications to send and receive data blocks to the NASCOM driver. The NASCOM interface also accepts history data for the purposes of simulation and testing. The various types of data blocks received by the NASCOM Interface include:

- Telemetry.
- NCC.
- Local TAC.
- Command echoes.
- Simulation data (history).

The different types of data blocks output includes:

- Command.
- NCC.
- Local TAC.

All input and output blocks are logged to the history file (via the history subsystem). The NASCOM Interface buffers all blocks received and routes them to the appropriate AP task. To output a block, the programmer will get a buffer, fill it with data, and then queue it for output to the NASCOM driver.

8.3.3.2.3 TAC Interface

The Telemetry and Command (TAC) Interface provides the application processor operator with remote control of the TAC.

8.3.3.2.4 Terminal Interface

The Terminal Interface provides communications between the AP applications and the operator terminals. This interface provides the following functions:

- Logging terminals into and out of the STOL environment.
- Reading STOL inputs from the operator terminals.
- Writing STOL prompts, STOL status messages, event updates, and page updates to the operator terminals.
- Bypassing the terminal interface for special requirements.

When using the on-line system, the user will always log into the STOL environment. After this point, all AP application software will receive input via STOL. The Terminal Interface provides various functions for terminal output of different messages and events.

The terminal display during an on-line session is partitioned into lines (or sets of lines) for input and different types of output. Colors are extensively used to differentiate meaning of different lines. Figure 8-5 illustrates the layout of the display.

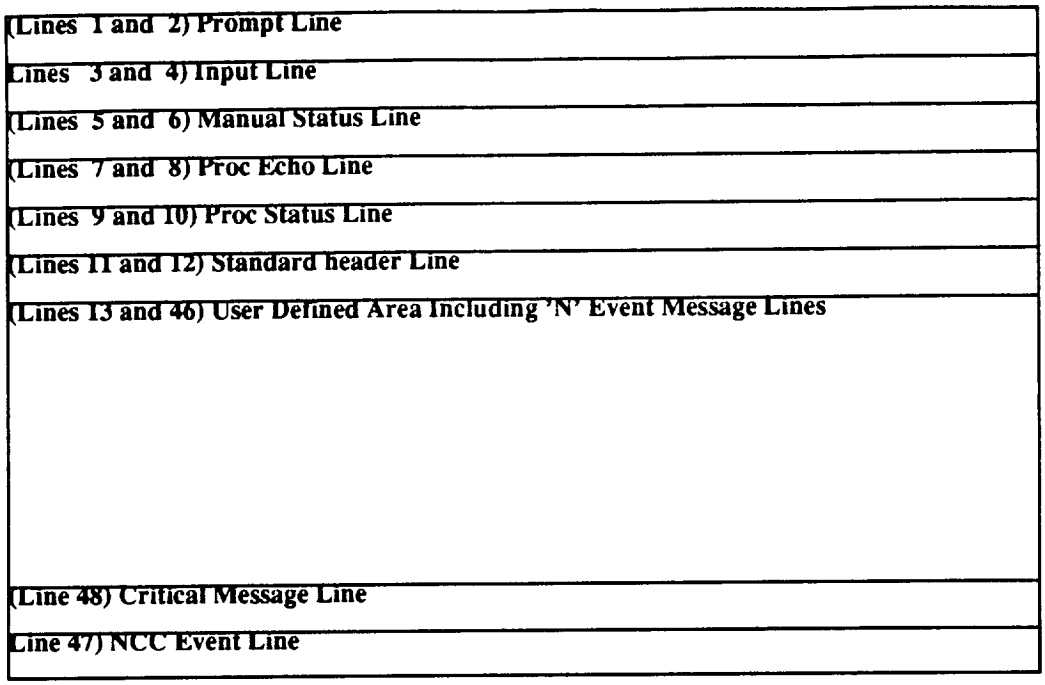


Figure 8-5 MAE Display Format

The Terminal Interface does not provide functions for page updates (lines 13-46). This mission-specific software must be provided by the mission developer.

8.3.3.2.5 GMT Interface

The GMT Interface communicates with the GMT device for the current time. This information is stored globally to allow access by all AP applications.

8.3.3.2.6 Device Configuration Interface

The Device Configuration Interface allows AP applications to obtain status and maintain the configuration of all AP peripherals including CRT's, line printers, tape devices, and stripchart recorder.

This interface also allows direct access to line printers. Functions are provided for opening, writing to, and closing the AP line printer.

8.3.3.3 Display Subsystem

The Display Subsystem provides library functions for event processing. Events are English text messages generated by applications running on the AP. This information is required to communicate system and satellite status to the AP operator. Event processing is performed by three tasks which supervise the following:

- Initialization of a central event queue (to which all events are sent).
- Transfer of event messages from the queue to buffers for appropriate event destination.
- Output of event buffers to event printer.

The Display Subsystem handles output of event messages to the static lines shown for the sample terminal display. Mission-specific software is responsible for output of events to the user defined

area and for relating page display to specific events. The Display Subsystem only supports event processing. This subsystem does not provide support for generation of display pages. This is the responsibility of the mission-specific software.

8.3.3.4 STOL Subsystem

The System Test and Operations Language (STOL) is a complete high-level language which allows the operator to control the application processor. STOL provides the required generic functionality and is the primary means whereby operators control a mission. Use of STOL eliminates the need for the operator to use the host operating system command line interface.

STOL statements are called "directives." Directives may be entered interactively, read from procedure files, or issued internally from applications. STOL directives are interpreted and executed as follows:

- Logs directive to events subsystem.
- Verifies authorization of the user initiating the directive(s).
- Performs syntax checking.
- Routes directive to appropriate applications task.

The operator is allowed to interactively edit STOL procedures using the system editor. STOL procedures provide a number of programming features, including:

- Up to 7 levels of procedure nesting.
- String argument passing and substitution.
- Access to data base variables (system and user).
- Branching.
- Conditional directive execution.
- Allows operator to control procedure execution.
- Procedure-unique directives, including proc/endproc, start, wait, go, goto, position, if, block if, step, ask, return, and killproc.

STOL includes a basic set of directives required for all missions. MAE allows new STOL directives to be added for mission-specific requirements. Mission-specific directives are created by:

- Building a "directive definition" which defines the directive syntax.
- Design and implementation of an on-line task to execute the directive.

8.3.3.5 NCC Subsystem

The Network Control Center (NCC) Subsystem allows messages to be sent to and received from the NCC. This subsystem allows operators to build outgoing NCC messages using definitions in the MAE database.

This subsystem processes incoming NCC messages by parsing parameters and storing the data system globals.

8.3.3.6 History Subsystem

The History Subsystem provides logging of mission data to a disk file. This subsystem creates a single physical file which is in turn divided into logical partitions for each satellite pass or event. The first block of the file is a directory which includes the starting position of each logical partition.

All NASCOM input and output blocks and event messages (including STOL inputs and status messages) are logged to the history file. This subsystem also allows mission-specific data to be logged to the history file. It is possible to define six additional types for history logging.

This subsystem provides several functions for maintenance of the history file. These include the following:

- Printing the history file directory.
- Compressing the file.
- Deleting a logical partition.
- Saving a logical partition to tape.
- Loading a logical partition from tape.

This subsystem does not provide either delog or playback capabilities. These functions must be provided by mission-specific software. Functions are provided however for reading data from the file.

8.3.3.7 Library Subsystem

The Library Subsystem provides general-purpose and MAE/mission interface support routines to the applications programmer. This subsystem consists of four object libraries which provide routines relating to the following functional areas:

- STOL.
- Events.
- History.
- Buffer pool.
- NCC interface.
- Device management.
- Network Interface Services (NIS) interface.
- Conversion.
- System service.
- Data base.
- File transfer (MODLAN).

8.3.3.8 Database Subsystem

The Database Subsystem allows definition of the different data used by the system. The data base definitions are contained in normal ASCII files. These raw data base files are input to the MAE data base generation software, which in turn creates operations data base files. The MAE data base generation also produces FORTRAN include files for use in programs. The Database Subsystem allows definition of the following types of data items:

- System globals - variables required for the system to operate. System globals are set and modified only by software.
- User globals - mission specific variables. These may be modified by STOL directives.
- NCC messages - definitions of all incoming and outgoing messages.

- MAE STOL operations classes and default assignments - used to perform protection checks on operator inputs and to assign classes to terminals.

8.3.4 Applicability to the Concept Executive

The MAE system is somewhat dated and therefore does not demonstrate usage of distributed processing and workstations. MAE also has a rather static terminal based user interface which is not useful for the more dynamic interface specified by the Concept Executive. There is also no discussion of how configuration management and security is provided.

The most important point about the MAE system is that it is a true executive. MAE offers an example of an executive implementation which includes a basic set of efficient functions which provide common support for requirements of different systems. The basic MAE system is then expanded to support the mission-specific requirements of each spacecraft. MAE provides an example of what is and is not considered within the scope of the executive. This functional scope can be applied to the Concept Executive.

Another interesting feature of MAE is the STOL language. This language completely replaces the native command line interface. All command level user interaction is input via STOL directives. Use of STOL provides an example of a custom language used to effectively support a control center environment. STOL is effective enough in fact that it will retained for use in the new TPOCC system described in a previous section.

Despite the example of the STOL language, the Concept Executive will specify use of the native command line interface.

8.4 HOSC PPS System

The Huntsville Operations Support Center (HOSC) Peripheral Processor System (PPS) uses a distributed processing concept to accommodate a wide range of Marshall Space Flight Center (MSFC) project requirements. The system's central processors and peripheral processors receive, process, and display the data necessary to conduct and control a payload mission.

8.4.1 Contact Point

Science and Engineering Directorate

Marshall Space Flight Center

8.4.2 Review Process

The review process for the HOSC PPS system involved examination of the following documents:

- HOSC Peripheral Processor System Overview, Volume I.
- HOSC Generic Peripheral Processor System User's Guide, Volume III: Display Utilities.
- HOSC Generic Peripheral Processor System User's Guide, Volume IV: User Special Computations.
- HOSC Generic Peripheral Processor System User's Guide, Volume V: On-Line Processing.
- MSFC Payload Operations Control Center Telemetry and Command Data Base Definition.
- MSFC Payload Operations Control Center (POCC) Capabilities Document.

Because the HOSC PPS is a system within the MSFC Payload Operations Control Center (POCC), SwRI reviewed the latter two documents to gain an understanding of the environment in which the PPS operates.

8.4.3 System Description

A major objective of the HOSC PPS is to provide a general purpose system useful to support multiple MSFC projects. The system provides a standard set of services designed and implemented specifically to allow the shared use of HOSC capabilities. The PPS is a general purpose, distributed processing system for processing and display of spacecraft telemetry and command data.

8.4.3.1 Operating Environment

The PPS is a part of the MSFC POCC. The POCC is an element of the Huntsville Operations Support Center (HOSC), and contains facilities and system resources to allow a payload user to monitor and control a payload/experiment during flight operations, preflight tests, verification, and simulations. The POCC provides resources which allow the user to:

- Monitor experiment functions and activities using down-linked digital, video, voice, and analog data sources.
- Correlate activities with shuttle-related parameters such as attitude and trajectory data.
- Send commands to the payload/experiment.
- Interface with the POCC systems from remote facilities for data stream transmission/receipt, computer file transfer, pre-mission training/display development, etc.
- Interface with the Mission Control Center (MCC).
- Coordinate with other operational personnel internal/external to the POCC.
- Interface with mission planning and other analysis computers external to the POCC.

The POCC is composed of the following major subsystems:

- Facilities - This includes the physical resources and utilities needed for preflight and flight support of a payload mission.
- Digital Data System - The digital data system provides the capability for reception, recording, processing, and distribution of digital data streams with the POCC. The digital data system includes the data acquisition and distribution system (DADS) and the peripheral processor system (PPS). These two systems provide the following capabilities:
  - External interfaces to the POCC data systems.
  - Receipt and transmission of digital data streams.
  - Processing of the Spacelab High-Rate Multiplexer data stream.
  - Recording and distribution of independent payload data streams (IPDS) and direct access channels (DACH).
  - Pre-processing and recording of the Orbiter operational downlink (OD) telemetry data stream.
  - Acquisition and distribution of input telemetry and non-telemetry data for monitoring by POCC users.
  - Definition, modification, and maintenance of displays.
  - Definition, modification, and maintenance of user special computations.
  - Display of data, update of forms, and execution of commands for uplink.

- File transfer among the PPS processors.
- Capability to uplink commands to payloads or experiments.
- Common use computations on downlink OD telemetry and user supplied mission planning data to generate displays distributed throughout the POCC.
- Receipt of trajectory, attitude, and state vector information from the MCC.

- Voice System - The voice system provides communications internal and external to the POCC including the capability to communicate with the payload crew onboard the Orbiter.
- Video System - The video system provides distribution and recording of downlink video from the Orbiter and from various other sources internal and external to the POCC.
- Analog System - The analog system provides distribution of downlink analog data from the Orbiter to user experimenter ground support equipment (EGSE) in the POCC.
- Timing System - The timing system provides distribution of timing in the POCC including distribution to user EGSE.

8.4.3.2 Hardware

The primary data system elements of the HOSC PPS are the central processors, peripheral processors, and user terminal devices. The central and peripheral processors are linked by an Ethernet local area network (LAN). The central processor is linked to external systems including the POCC Data Acquisition/Distribution System by the POCC Data Distribution Switch.

The peripheral processor system (PPS) includes processors designated as central processors (CP) and peripheral processors (PP). The central processor receives and stores the data and supplies it to the peripheral processor user upon request. The peripheral processor performs data conversions, user special computations, and display generation.

8.4.3.2.1 External Interfaces

The HOSC PPS receives telemetry data and transmits command and ancillary data through the POCC Data Distribution Switch. The PPS also interfaces with the High Data Rate System (HDRS) which provides the capability to record, process, and route data streams down-linked to the POCC via the POCC Data Distribution Switch. The HDRS receives real-time down-linked data from the NASCOM system including Spacelab High Rate Multiplexer Data, Direct Access Channels, and Independent Payload Data Streams.

8.4.3.2.2 Central Processor

The main functions of the Central Processor System are to:

- Receive, log, and distribute telemetry measurements upon request.
- Perform common use computations on incoming data that are of interest to multiple users.
- Maintain a near real-time data base of up to 24 hours of data stored on disk.
- Provide data transmission services.
- Log data transactions.
- Maintain the telemetry reference data base.

The central processor (CP) acquires and stores telemetry and ancillary data in main memory buffers for real-time access. This data is accessed in servicing requests for the PP's and in executing common use computations.

Data is validated upon receipt, logged by the system, and made available for display through the peripheral processors. The data is decommutated through the use of the reference data base and distributed on a parameter basis to the peripheral processors. Parameters are also decommutated and provided to a strip chart PP for output to strip chart recorders.

Common use computations (CUC) are performed on the input data. These CUC's provide special processing functions not available through standard conversion yet of general interest.

Exception monitoring is a CUC that performs limit/expected state sensing on specially identified parameters. The command history delog CUC generates command delog reports from command records logged to the CP composite system log. These and other CUC's are performed on the CP to log and validate data received.

The CP provides users with the capability to recall and process near real-time data. The CP also transmits data to the PP's for user display and processing. Parameters are provided to the strip chart recorder PP by the CP. The CP also provides up to four real-time and four playback experimenter ground support equipment (EGSE) subsets to user EGSE.

The CP also maintains a system log which is a composite recording of significant events created during operation. This log includes POCC commands, exception monitoring advisories, and other system advisories.

The central processor computer systems contain the following hardware components:

- Perkin Elmer 3280 processor.
- OS/32 Operating System V6.1.
- FORTRAN 77.
- Four megabytes of memory.
- Line Printer.
- Two 300 megabyte disks.
- Three 600 megabyte disks.
- Two 1600 BPI tape drives.
- Four 6250 BPI tape drives.

8.4.3.2.3 Peripheral Processors

The peripheral processor provides the user interface for building displays and special computations; for viewing commands, telemetry, and status information; and for modifying and up-linking commands. Two modes are available to the user:

- Off-line mode to generate or modify/maintain user-built displays and special computations.
- On-line mode used to process and view telemetry data, and to modify and execute commands.

There are three functional types of peripheral processors within the PPS: the command/display PP, the strip chart PP, and the user PP's.

The command/display PP provides the following storage and control functions for the PPS:

- Centralized storage of system and user directories which contain displays, command forms and user special computations.
- Centralized storage for the command data base.
- Magnetic tape drive for transporting tapes between compatible systems and for backup of system and user directories for permanent storage.
- High speed line printer for printing NRT reports and tabular formatted displays and command forms.

The hardware complement for the command/display PP includes the following:

- VAX 8250.
- VMS operating system.
- 8 megabytes memory.
- 445 megabytes disk space.
- Floating point accelerator.
- RX 50 dual floppy disk.
- 1 TA81 tape drive.
- 4 work station video display terminals.
- LN03 laser printer with graphics.

The strip chart PP receives data for strip charting from the CP. This processor performs the following functions:

- Receives and processes data for strip chart recording.
- Provides a five-step calibration output to the strip chart recorder in response to user requests.

The hardware complement for the strip chart PP includes the following:

- VAX 11/730.
- VMS operating system.
- FORTRAN 77.
- 3 megabytes memory.
- RA80 disk.
- 2 TU80 tape drives.
- RX02 floppy disk.
- B1000 data products printer.
- 4 work station video display terminals.
- 64 channel digital to analog converter.
- Eight 8-channel strip chart recorders.

The user PP's are microcomputers that are interfaced to user POCC terminals. The user PP's provide the following:

- Real-time storage for user displays and special computations for on-line processing operations.
- Storage of updated command forms, display pages, and ground support equipment.
- Relemetry processing functions associated with display requests and user special computation execution.
- Interface with the command/display PP for update of the command data base.
- Interface with the CP for update of the reference telemetry data base.
- Interface with the CP, strip chart PP, and command/display PP for system control and monitoring.

The hardware complement for the user PP's includes the following:

- Microvax II.
- VMS Operating System.
- 3 megabytes memory.
- 71 megabytes disk space.
- Floating point accelerator.

- 4 work station video display terminals.
- LA210 letter printer with graphics.

8.4.3.3 PPS Users

User accounts to support a specific activity are established on a mission to mission basis and on predefined peripheral processors (PP). User accounts are established for activities, not for individuals. Specific items throughout the system and support activities are associated with each user. User accounts provide privacy for the following:

- Uduser defined displays.
- Command update forms.
- User special computations.
- User defined private data.

The general user from the payload community or vehicle systems is assigned a 2 digit numeric code used for identification. Several unique operation positions are identified for the PP system and have been assigned alphabetic user ID's to distinguish their special roles:

- PC - Payload Command Control User.
- CC - Configuration Control User.
- HR - High Data Rate System Control User.
- OC - Operations Controller.

A four digit mission ID code is assigned to each mission and combined with the user ID to create an account ID. Support activities are associated with the mission ID. The software system is then

generated and configured to support each activity. This allows the system to maintain a separate configuration of mission-specific software, data bases, and common and user special computations.

8.4.3.3.1 Payload Command Control User

The PC position controls the POCC command uplink functions by enabling/disabling uplink, commands, and users. The PC user may only access payload commands to perform these functions; however, the PC user may set his user ID to a numeric user ID and have access to commands in that user account.

8.4.3.3.2 Configuration Control User

The CC user controls the allocation of system resources during support activities. The CC user controls the exception monitor capability which performs limit/expected state sensing of a limited number of critical parameters by common use computation residing in a central processor. The CC user responsibilities also include control over reference data base (RDB) changes, strip chart recorder assignments, and writing non-telemetry data to the central processor.

8.4.3.3.3 High Data Rate System Control User

The HR user controls the capability of a user to send commands to the HDRS. The HR user may:

- Enable or disable HDRS command transmission for all users or a specific user.
- Enable or disable checking of the hardware allocation table when commands are sent to the HDRS.

8.4.3.3.4 Operations Controller User

The OC user may set his user ID to a legal numeric user ID and have access to commands in that user account.

8.4.3.3.5 User Access

The user may gain access to the peripheral processor only at the times the account is enabled. The user is limited to input as specified by the software, such as entry selections that are menu defined. Escape to the operating system is prohibited. When the user selects the Exit entry from the menu, the session is terminated and the user is logged off the system.

8.4.3.4 Reference Data Base

The PPS maintains reference databases which contain telemetry data, command information and special parameters.

The telemetry reference data base is maintained on the central processor and contains the processed result of the telemetry file from the mission manager's data tape. This information satisfies three functional requirements:

- Retrieval for display, editing, and replacement of parameter information.
- Retrieval for display of telemetry and pseudo-telemetry data.
- Updating of pseudo-telemetry data items.

The telemetry data base report includes the measurement stimulation identification (MSID), a description, engineering units, calibration information, data type and length, and the user ID. Other reports available upon request to the configuration control user contain information specific to

- The strip chart recorders.
- The experimenter ground support equipment (EGSE).
- Exception monitors.

The command reference data base resides on the POCC Command and Display System which is a peripheral processor within the PPS. This data base contains the commands associated with each user account. Users develop and maintain command chains to support mission activity. A complete list of all commands and their pertinent specifications provided by the mission manager are stored in the command reference data base.

The command processing software generates data base reports which support the command activity of the general user. Information provided to the configuration control user includes:

- A listing of command mnemonic names sorted by user ID and how many each user ID has.
- A report of mnemonic name compilation process.
- A report of the shell array compilation process which contains the mnemonic name and the number of data cards.
- A decoded formatted list of real-time storage array data from the compilation process of the modifiable and/or non-modifiable commands.

Selected parameters are identified in the data base for specific processes including exception monitoring, strip chart recording, and the experimenter ground support equipment. System status parameters which reflect the configuration and status of the PPS are also included in the data base.

8.4.3.5 PPS Functions

The peripheral processor software is the user interface for building displays and special computations; for viewing commands, telemetry, and status information; and for modifying and up-linking commands. The peripheral processor software includes the following subsystems:

- On-Line Processing.
- Display Generation/Modification.
- User Special Computation Utilities.

The peripheral processor software also provides the capability to initiate commands to experiments and Spacelab payloads onboard the Orbiter. Commands are validated by the PPS and transmitted to MCC which provides a command acceptance pattern back to the POCC. After validation at the MCC, the commands are transmitted to the Orbiter. Control provided by the command processing software includes:

- Capability to enable/disable any user or the system.
- Command safing for critical commands.
- Tracking of command activity of each user and the total system.
- Capability to decode, monitor, and route response messages from Mission Control Center.

8.4.3.5.1 On-Line Processing

The on-line processing mode provides the access to mission data from the CP through the LAN. This is the operational mode for active mission and test support. It provides the capability to view data on user-built displays or system-provided displays. In this mode a display is on the screen at all times.

Software to support on-line processing functions executes in both the CP and the PP. All user-specific processing such as engineering unit conversion, limit sensing, NRT report generation, command modification, user special computations, conversion and formatting for displays is performed in the PP.

The on-line processing software allows the user to view data on displays that he has built or that are provided by the system. Data items are referenced by MSID. The user may also view command status information and uplink commands which are enabled. Streams of data are provided to the peripheral process by the central processor and are available in four modes:

- Real Time - users can look at data as it is coming into the POCC simultaneously with the receipt of the data.
- Near Real Time - users request time slices from data stored on disk; up to 24 hours of data are available on disk.
- Playback 1 - users look at data from a high data rate recorder/payload recorder dump.
- Playback 2 - users may make a request for playback of a previously recorded time slice of data.

The display layout includes a scratch-pad line from which the user may communicate with the on-line processing software using predefined instructions or directives. Limit sensing is performed by the peripheral processor on a display basis and only for the values displayed. The user defines warning and red-line limits at the time the display is built.

8.4.3.5.2 Display Generation/Modification

Display generation utilities are provided to the user to build and maintain free form displays during the mission activity period, share those displays with other users, and view the displays in an on-line operation. The following functions are provided by the display utility software:

- Build or modify composite file.
- Build or modify a display.
- Show a directory of files.
- Download files from the Command Display System (CDS).
- Save files on CDS.
- Delete user files.
- Store files in shareable library.
- Copy local or shareable files.
- Rename files.
- Show updatable display data pages.
- Print display.

The composite edit function enables the user to build files containing composites which are the building block for schematic displays. Composites consist of one or more objects which may be colored and/or shaded. An object may be a circle, box, or vector figure. These composites are used in the display build process.

The display edit function enables the user to define the 24-line display layout. Lines 1, 22, 23, and 24 contain information denoting the status of the PPS and user interaction with the system. These lines contain the current GMT, current MET, a user defined display title, data mode for the display, display name, date of last update, space for exception monitoring and user advisories, the scratch-pad line, and a status line for user inputs.

The user inserts display information in lines 2 through 21. Using graph mode, the user defines graphs to be plotted on the display. Using the layout mode, the user defines the free form data displayed in areas not occupied by graphs. Using the field definition mode, the user specifies the MSID for each measurement to be displayed or updated.

Using the recall definition mode, the user specifies the contents of recall fields defined in the layout mode. Up to 80 characters of text may be associated with each recall field. During operations, this text can be recalled to the scratch-pad line.

These utilities operate in an off-line mode and do not receive data from the central processor. Display definition capabilities include the following:

- Specification of up to 100 data fields.
- Specification of up to 50 recall fields.
- User definition of lines 2 through 21 of the 24 line display.
- Identification as single shot (display updated only once) or cyclic display.
- Refresh rates between once per second and once per minute for cyclic displays.

8.4.3.5.3 User Special Computation Utilities

The special computation software is used to satisfy telemetry processing situations which require a computation sequence involving one or more parameters. This software allows the user to create, maintain, and execute analysis programs needed to perform mission support. Special computations execute on the peripheral processor and have access to real time or near real time data. Each computed parameter is available for display.

Special computations are developed in FORTRAN using the standard DEC editor. In addition to the standard DEC system services (excluding services to create and delete processes), a library of support routines is provided. The library routines provide the following capabilities:

- Initialization - The initialization routines are used to create and initialize telemetry, non-telemetry, and constant data for input to the PP from global common and the CP, and output to global common and the CP.
- Calibration - The calibration file routines input calibration data for each of the specified parameters from the calibration data base resident on the CP.
- Engineering Unit/State Code Conversion - The engineering unit/state code conversion routines provide the capability to convert raw telemetry data into its appropriate data format. Parameters having line segment or polynomial coefficient type of calibration

are converted into engineering units. Parameters having state code type of calibration are converted into character descriptors.

- Data Input - The data input routines input the current value of constants and non-telemetry parameters from global common and telemetry or non-telemetry parameters from the CP.
- Data Output - The data output routines output the current value of non-telemetry parameters from the special computation to global common and the CP.
- VT 100 - The VT 100 routines allow the creation and manipulation of text information on any terminal operating in VT 100 mode.
- Miscellaneous routines provide various other special computation system capabilities.

The user special computations are compiled and linked by invoking menu options provided by the PP Special Computation Utilities Menu. This menu also provides access to file maintenance utilities and other development tools including the following:

- Directory of Special Computations.
- Move Special Computations to PP (from permanent storage on the Command Display System).
- PP File Processing:
  - Copy.
  - Delete.
  - Edit.
  - Print.
  - Purge.
  - Rename.
  - Type.
- Compile Special Computations.
- Link Special Computation.
- Special Computation Form Processing:
  - Create/modify special computation input forms.
  - Create/modify special computation output forms.
  - Eit special computation forms logic file.
  - Execute special computation forms precompiler.
- Run Special Computation With Display.
- Save Special Computations on Command Display System (permanent storage for special computations is maintained on the Command Display System).
- Delete Special Computations from Command Display System (CDS).

The Special Computation Form Processing Software is a type of application generator which provides an alternative method of developing a special computation. This software is designed to guide the user throughout the task of creating or modifying a form. Using this software, the user

- Creates an input form which defines the parameters to be input.
- Creates a logic file which defines the basic structure of the special computation.

- Creates an output form to define the computation's output to global common and/or the CP.

All required inputs by the user are displayed and the user is prompted for input from the keyboard. If a field definition can be supplied by the software or by reading the data base, the user is not prompted to supply it.

Permanent storage for user special computations is maintained on the CDS. The routines are placed in a library and access is controlled via user ID/password. User special computations may be placed in a shared library to provide access to multiple users. When a computation is copied to the shared library, it is not removed from the user specific library.

8.4.3.6 Configuration Control

Configuration control is maintained via the reference data bases and the following operation positions:

- PC - Payload Command Control User.
- CC- Configuration control User.
- HR - High Data Rate System Control User.

The commands, displays, and user special computations are stored in the reference data bases on the CP and the Command/Display PP. The PC, CC, and HR control users enable and disable user and system capabilities.

Use of command services is controlled by the PC user. This user enables/disables the command system, a particular user, and individual commands for uplink. The ability to modify commands is dependent upon the definition of the command in the command data base. Commands may be defined as follows:

- Predefined commands are totally defined in structure and content and cannot be changed during support.
- Modifiable commands have a predefined structure but the data content of the command may be changed during support.
- Variable length modifiable commands are defined during flight.

8.4.4 Applicability to Concept Executive

The PPS incorporates several design concepts which are applicable to the Concept executive. These concepts and their applicability are discussed within the following framework:

- Configuration control.
- Distributed processing.
- User interface.

The PPS maintains configuration control utilizing several methods. First, special user positions are defined and assigned areas of responsibility. These users enable/disable system and user capabilities. Second, centralized storage is provided for user developed mission support software. Since access to this storage is controlled by the special user positions, a second level of control is maintained by restricting pre-mission updates and on-line access of the storage. Third, user accounts are assigned for each mission.

The PPS distributes processing between central processors and peripheral processors. The central processors receive and store the data and supply it to the peripheral processor. The peripheral processor performs data conversions, user special computations, and display generation. Process allocation between the peripheral processors is further divided between user peripheral processors (PP), strip chart PP's, and the command/display PP.

The PPS provides the user with display generation utilities to allow the user to develop custom display layouts for user specific mission support software. In addition to the display layout, the user defines the information to be displayed including telemetry data, command information, and status information. This gives the user a considerable level of flexibility and control over the mission support activity.

However, the PPS does not appear to utilize available standards and seems to be tightly coupled with its hardware/software platform. The user interface does not utilize existing workstation technology.

8.5 IRAF System

The IRAF system is a publicly available software system available from the National Optical Astronomy Observatories (NOAO). This system is currently in use at a number of commercial and government sites. The primary users of the IRAF system are astronomers who gather and manipulate images taken from large telescopes. It is currently being used at a number of observatories and is the primary system used for the Space Telescope project.

IRAF is a large and complex system with well over 100,000 lines of code. It has been publicly used for several years and runs on a variety of UNIX and VMS-based hosts.

Although designed for astronomers and for image manipulation, the IRAF system has sufficient capability to support many applications and users. While not a distributed software executive, it does provide enough functionality from which such a system could be developed. In addition, the IRAF system provides unique solutions to the problems of portability and high-level language/OS interfaces. Many of the requirements of a distributed software executive are also met by the functionality inherent in this system.

8.5.1 Contact Point

The IRAF system is distributed by the National Optical Astronomy Observatories. It is publicly available to interested organizations. To obtain a copy of the software and documentation, contact:

IRAF

National Optical Astronomy Observatories (NOAO)

P.O. Box 26732, Tucson, Arizona, 85726

8.5.2 Review Process

The review process for the IRAF system involved installation and use of the 2.8 version of the software on a Sun 4/150 system. In addition, the following documents were reviewed:

- IRAF User Handbook Volume 1A - IRAF System.
- IRAF User Handbook Volume 2B - User's Guides/KPNO Cookbooks.

8.5.3 System Description

The IRAF system was designed to allow users to gather and manipulate image data. It provides all the necessary functions for a scientist to effectively perform this function. It provides a user interface and large set of applications which are sufficient for the majority of user requirements. In addition, it provides a high-level programming language and a virtual operating system. This allows users to develop new applications to support future requirements. By defining a new language and a portable operating system, it is possible to write applications which are completely portable on any host. The task of porting the IRAF system involves developing the relatively small set of support functions which in turn support the virtual operating system.

To describe the IRAF system, it is best to examine it from the top down (from the primary user interface down to the low-level host-dependent functions). The five basic functional components (as described in this document) are as follows:

- Command Line (CL) - the command line is an interactive interpreter similar in function and syntax to the normal UNIX C shell. The CL allows a user to enter commands and provides a programming language which allows scripts to be developed.
- Applications - an application is a program or script which is normally executed via the Command Line. Applications include those for system, math, image processing, and other functions.
- Subset Preprocessor (SPP) - this is a high-level language provided by IRAF. This high-level language is preprocessed by the IRAF compiler into host FORTRAN, which is then compiled into an executable application. It is this language that the vast majority of IRAF applications and functions are written in. Note that IRAF also provides a small set of C and FORTRAN callable functions.
- Virtual Operating System (VOS) - the VOS is a complete operating system which resides above the host operating system (UNIX or VMS). It provides a complete set of utilities and functions and allows a programmer to develop applications which are independent of the underlying host. Note that the VOS itself is written in SPP.
- Host System Interface (HSI) - the HSI is the interface which implements the VOS and allows execution on the host. This is the only part of the system which is host dependent.

Each of the components described above is similar to a component in a typical UNIX development environment. Figure 8-6 illustrates this relationship.

IRAF Component	Equivalent in UNIX
Command Language	C or Bourne Shell
IRAF Applications	UNIX Commands
Subset Preprocessor	C and FORTRAN
Virtual Operating System	UNIX and Libraries
Host System Interface	N/A

Figure 8-6 Comparison of IRAF to UNIX

In a typical IRAF session, the user will use the CL to execute a variety of processes. A process may be a CL script, an IRAF application built in the SPP language, or a "foreign" process which is part of the host operating system. In IRAF, a process is normally grouped into what is called a "package." A package is simply a collection of tasks which have a similar function, such as system oriented or image processing. To execute a process from a given package, the user will first load the package. At this time all processes which are part of the package will be read into what IRAF calls its "process cache." Once in the cache, a process will execute very quickly as it is already memory resident. This makes execution of commands much faster, especially small interactive commands for which initialization is normally the major time concern.

Note that a package physically exists as a UNIX process. Executing tasks from the package consists of loading the package (running the process) and initiating tasks within it (IPC initiation of functions).

8.5.3.1 The Command Language (CL)

The command language (CL) is the user's interface to the IRAF system. The CL is very similar in use and function to the familiar UNIX C shell. Some of the features and functions of the CL are as follows:

- Provides a uniform environment on all host systems.
- Structure for organization and extensibility.
- Menus and extensive on-line help facilities.
- Command syntax similar to UNIX C shell.
- I/O redirection and pipes; aggregate commands.
- Match abbreviations for task and parameter names.
- Local and global parameters, hidden parameters.
- Access to host system; foreign task interface.
- Set editor; command history editor (edt, emacs, vi).
- Job submission (including queuing).

- Facility for recording all task invocations.
- Graphics and image display cursor mode facilities.
- Filename facility; unix style pathnames to files.
- Procedures, C style expressions and control constructs.
- Simple escape mechanism for direct access to host commands.

From the CL, the user will select tasks from different packages. Packages are organized and grouped by function, such as system, image processing, etc. The IRAF CL defines the following four types of tasks:

- Built-in tasks - these are simple functions which are inherent to the CL itself. These are similar to intrinsic functions available in the UNIX C and Bourne shells.
- Script tasks - these are tasks written in the command language. They are similar to UNIX C and Bourne shell scripts.
- Compiled tasks - these are tasks written in a compiled language and executing out of the process cache. These are most commonly applications written in the SPP programming language.
- Foreign tasks - these are host programs or scripts which are run directly on the host system.

The CL is the primary user interface to the IRAF system (just as the C shell is to UNIX). It provides a great deal of flexibility and is comparable in usability to the UNIX shell. It is beyond the scope of this document to compare the two environments on a detailed level. Suffice to say that the CL would be a comfortable environment for either a novice or experienced UNIX user to work.

One feature of note provided by the CL is the ability to set what are called "hidden" parameters. These task parameters are similar to familiar defaults in UNIX commands (such as whether "rm" prompts you or "ls" lists "." files. The CL provides reasonable defaults and allows the user to permanently affect the defaults (he may also temporarily override them via the command line). This is a simple operation in which a field-based screen editor is provided. Note that this is possible in the UNIX C shell via aliasing, but this method is more cumbersome.

8.5.3.2 Application Software

Application software consists of the various packages which are provided with the IRAF system. These include system packages and scientific reduction and analysis packages. The large number of packages and tasks is one feature which makes the IRAF system so powerful. This is similar to the UNIX system in which it is the large number of commands which actually provide much of the power of the system.

Some of the different packages and the tasks provided by the IRAF system include the following:

- DATAIO - I/O to various types of devices:
  - mtexamine - Examine the structure of a magnetic tape.
  - rcamera - Convert a Forth/Camera image into an IRAF image.
  - reblock - Copy a binary file, optionally reblocking.
  - wcardimage - Convert text files to cardimage files.
  - pdsread - Convert a PDS image into an IRAF image.
  - rcardimage - Convert a cardimage file into a text file.
  - rfits - Convert a FITS image into an IRAF image.

- IMAGES - Generic image processing functions:
  - imcopy - Copy an image.
  - imdelete - Delete an image.
  - imheader - Print an image header.
  - imhistogram - Compute image histogram.
- IMAGES.TV - Image processing specific to TV:
  - v - Image display load and control package.
  - blink - Blink two frames.
  - display - Load an image or image section into the display.
  - erase - Erase an image frame.
  - frame - Select the frame to be displayed.
  - lumatch - Match the lookup tables of two frames.
  - monochrome - Select monochrome enhancement.
  - pseudocolor - Select pseudocolor enhancement.
  - rgb - Select true color mode (red, green, and blue frames).
  - window - Adjust the contrast and dc offset of the current frame.
  - zoom - Zoom in on the image (change magnification).
- LANGUAGE - Functions intrinsic to the Command Language:
  - access - Test if a file exists.
  - bye - Exit a task or package.
  - cache - Cache parameter files, or print the current cache list.
  - chdir - Change the current working directory.
  - cl - Execute commands from the standard input.
  - defpac - Test if a package is defined.
  - defpar - Test if a parameter is defined.
  - deftask - Test if a task is defined.
  - envget - Get the string value of an environment variable.
  - error - Print error code and message and abort.
  - fprint - Format and print a line into a parameter.
  - fscan - Scan a list (formatted input).
  - history - Print the last few commands entered.
  - keep - Make recent set, task, etc. declarations permanent.
  - kill - Kill a background job.
  - lparam - List the parameters of a task.
  - mktemp - Make a temporary (unique) file name.
  - package - Define a new package, or print the current package names.
  - print - Format and print a line on the standard output.
  - radix - Print a number in the given radix.
  - redefine - Redefine a task.
  - scan - Scan the standard input (formatted input).
  - service - Service a query from a background job.
  - set - Set an environment variable, or print environment.
  - substr - Extract a substring from a string.
  - task - Define a new task.
  - unlearn - Restore the default parameters for a task or package.
  - update - Update a task's parameters (flush to disk).
  - version - Print the revision date of the CL.
  - wait - Wait for all background jobs to complete.

- LISTS - Functions to process lists of values:
  - average - Compute the mean and standard deviation of a list.
  - gcursor - Read the graphics cursor.
  - imcursor - Read the image display cursor.
  - table - Format a list of words into a table.
  - tokens - Break a file up into a stream of tokens.
  - unique - Delete redundant elements from a list.
  - words - Break a file up into a stream of words.
- PLOT - Generic plotting functions:
  - contour - Make a contour plot of an image.
  - graph - Graph one or more image sections or lists.
  - pcol - Plot a column of an image.
  - pcols - Plot the average of a range of image columns.
  - prow - Plot a line (row) of an image.
  - prows - Plot the average of a range of image lines.
  - surface - Make a surface plot of an image.
- SOFTOOLS - Software development tools:
  - make - Table driven utility for maintaining programs.
  - mklib - Make or update a library.
  - mkmanpage - Make and edit a new manual page.
  - xcompile - Compile and/or link a program.
  - yacc - Build an SPP language parser.
- SYSTEM - System functions:
  - allocate - Allocate a device.
  - beep - Beep the terminal.
  - bugmail - Print/Post bug reports, complaints, suggestions.
  - clear - Clear the terminal screen.
  - concatenate - Concatenate a list of files to the standard output.
  - copy - Copy a file, or copy a list of files to a directory.
  - count - Count the number of lines, words, and characters in a file.
  - deallocate - Deallocate a previously allocated device.
  - delete - Delete a file.
  - devstatus - Print the status of a device (mta, mtb, tty, ...).
  - directory - List files in a directory.
  - diskspace - Show how much diskspace is available.
  - edit - Edit a file.
  - files - Expand a file template into a list of files.
  - head - Print the first few lines of a file.
  - help - Print on-line documentation.
  - lprint - Print a file on the line printer device.
  - match - Print all lines in a file that match a pattern.
  - news - Page through the system news file.
  - page - Page through a file.
  - pathnames - Expand a file template into a list of OS pathnames.
  - protect - Protect a file from deletion.
  - rename - Rename a file.
  - revisions - Print/Post a revision notice for a package.
  - rewind - Rewind a device.

- sort - Sort a text file.
- spy - Show processor status.
- stty - Show/Set terminal characteristics.
- tail - Print the last few lines of a file.
- tee - Tee the standard output into a file.
- time - Print the current time and date.
- type - Type a file on the standard output.
- unprotect - Remove delete protection from a file.

• UTILITIES - Miscellaneous utilities:

- airmass - Compute the airmass at a given elevation above the horizon.
- ccdtime- Compute time required to observe star of given magnitude.
- entab - Replace blanks with tabs and blanks.
- lcase - Convert a file to lower case.
- precess - Precess a list of astronomical coordinates.
- translit - Replace or delete specified characters in a file.
- ucase - Convert a file to upper case.
- urand - Uniform random number generator.

This long list of applications is only the core set of functions provided by IRAF. It is also not necessarily exhaustive, as there have been updates made to the system. This list does not include the set of packages provided by the NOAO, which are a set of functions useful for specific astronomy requirements. There are also a variety of third-party packages available from existing IRAF sites. The fact that IRAF applications (SPP developed) are fully portable makes it possible to share code from other locations.

8.5.3.3 The Subset Preprocessor (SPP) Language

The subset preprocessor language is the primary programming interface to the IRAF system. It is a general purpose language modeled after C, but which is implemented as a FORTRAN preprocessor. In other words, the user programs in the SPP language (much like in normal C) then runs the SPP compiler which translates the SPP code into FORTRAN, which is in turn compiled into executable code for the host.

The SPP language most closely resembles C, but includes additional support for more complex mathematical operations. A good way to obtain a "feel" for the SPP language is to examine a short program using it. Figure 8-7 presents a program called "page," which is similar to the "more" command present in UNIX. This program is part of the system package, which provides commonly used commands.

```
# Copyright(c) 1986 Association of Universities for Research in Astronomy Inc.

include<fset.h>

# PAGE -- Display a text file or files on the standard output (the user
# terminal) one screen at a time, pausing after each screen has been filled.
# The program is keystroke driven in raw mode, and currently recognizes the
# keystrokes defined above.

procedure t_page()

bool        redirin
pointer     sp, device, prompt, files
int         map_cc, clear_screen, first_page

bool        clgetb()
intf        stati(), clgeti(), btoi()

begin
  call smark (sp)
  call salloc (device, SZ_FNAME, TY_CHAR)
  call salloc (prompt, SZ_FNAME, TY_CHAR)
  call salloc (files, SZ_LINE, TY_CHAR)
  redirin = (fstati (STDIN, F_REDIR) == YES)
  if (redirin)
    call strcpy ("STDIN", Memc[files], SZ_LINE)
  else
    call clgstr ("files", Memc[files], SZ_LINE)

  map_cc = btoi (clgetb ("map_cc"))
  clear_screen = btoi (clgetb ("clear_screen"))
  first_page = clgeti ("first_page")
  call clgstr ("prompt", Memc[prompt], SZ_FNAME)
  call clgstr ("device", Memc[device], SZ_FNAME)

  call xpagefiles (Memc[files], Memc[device], Memc[prompt],
    first_page, clear_screen, map_cc)

  call sfree (sp)
end
```

Figure 8-7 Sample SPP Program

As the example shows, the SPP language has a large number of similarities to both C and UNIX, including similar constructs and function calls.

As with any programming language, it is only as powerful as the available support libraries. In addition to the functions provided by the virtual operating system itself (similar to UNIX system calls), the following general-purpose, graphics, and mathematical support libraries are provided:

- bev - Bevington routine package.
- curfit - 1-dimensional curve fitting package.

- deboor - DeBoor spline package.
- GKS - IRAF GKS emulator package.
- surfit - Surface fitting (regular grid) package.
- gsurfit - Surface fitting (irregular grid) package.
- iminterp - Image interpolation package.
- llsq - Lawson's and Hanson's linear least squares package.
- ncar - NCAR graphics (GKS version) package.
- nspp - Old NCAR system plot package.
- xtools - General tools library for SPP applications programs.

The major advantage of the SPP language is that it can be used to write truly portable applications. The combination of the language, the support libraries, and the underlying virtual operating system provides a programming environment which is very conducive to development of portable applications. While the same is possible with UNIX, C (or FORTRAN), and standard support libraries (X, GKS, etc.), it is much more difficult due to the dependence on correct implementation of standards and requires programmers to contentiously avoid use of non-standard features.

Since the SPP preprocessor translates SPP into FORTRAN, it can generate the appropriate code for the given host compiler and use host-specific optimization techniques as necessary.

In addition to the SPP language interface, the IRAF system provides a small subset of C and FORTRAN callable functions. However, the interfaces are incomplete and do not provide full (or even close) access to the features of the virtual operating system. These interfaces exist to allow large existing applications access to IRAF data. The design of the IRAF system is such that in order to develop any significant application, it is necessary to learn the SPP and the virtual operating system.

8.5.3.4 Virtual Operating System (VOS)

The virtual operating system provided by the IRAF system is a complete programming environment. It is similar in concept to the UNIX operating system, in that it provides all the required functionality to develop a variety of applications. The VOS also provides programming functionality which is useful in scientific environments, such as those requiring image processing, high-level mathematics, or interactive graphics.

The IRAF VOS describes the functionality provided in several areas. These include the following:

- CLIO - command language I/O (get/put parameters to the CL).
- DBIO - database I/O.
- ETC - exception handling, process control, symbol tables, etc.
- FIO - file I/O.
- FMTIO - formatted I/O (encode/decode, print/scan).
- GIO - graphics I/O (both vector graphics and image display access).
- IMIO - image I/O (access to bulk data arrays on disk).
- KI - kernel interface (network communications).
- LIBCUNIX - stdio emulation, C binding for the VOS, used by the CL.

- MEMIO - memory management, dynamic memory allocation.
- MTIO - magtape I/O.
- OSB - bit and byte primitives.
- TTY - terminal control.
- VOPS - vector operators (array processing).

It is beyond the scope of this document to examine the details of these functional components. Suffice to say that the VOS is similar in concept and function to the UNIX operating system and is capable of supporting development of large and complicated applications.

It is important to note that the VOS is not a series of processes which run on top of the UNIX operating system. Rather, the functionality provided by the VOS is ingrained in the executable program created via the normal load step. The basic structure of a program is illustrated in Figure 8-8.

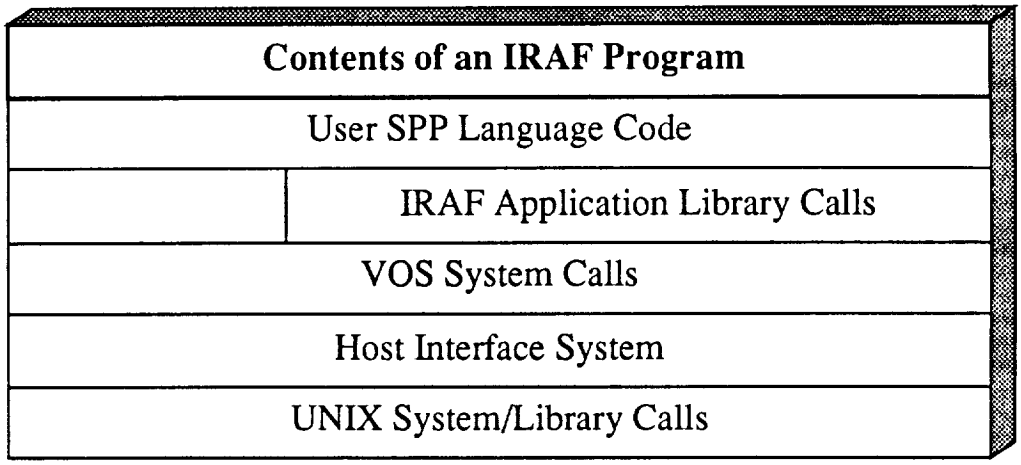


Figure 8-8 Contents of an IRAF Program

The IRAF VOS is entirely written in the SPP language. As such, it is another part of the system which is entirely portable. This is important as the functionality in the VOS is complex and porting it to a new system would be difficult if it were implemented in a host-specific language.

8.5.3.5 Host System Interface (HSI)

The host system interface is the interface between the portable components of the IRAF system (VOS and applications) and the host operating system itself (UNIX or VMS). The HSI contains ALL of the host-dependent or potentially host-dependent code. When the IRAF system is ported to a new host, it is the HSI which actually must be ported. Once complete, the entire VOS and all applications will be completely portable. The HSI consists of the following major components:

- IRAF kernel - consists of a library of approximately 50 FORTRAN callable functions. These are currently written in C, but may be in any language as long as they are callable by FORTRAN (necessary due to SPP). The kernel is a small set of functions which provides host-dependent services for the VOS. This kernel is well-contained and may be implemented according to specifications without any knowledge of the higher level software.

- Bootstrap utilities - these are required to compile and maintain the remainder of the IRAF system. These utilities are required to generate the IRAF system and are therefore written in the host language. They consist of the SPP compiler, tape read/write utilities, YACC (Yet Another Compiler Compiler) utility to build the SPP compiler, and other functions.

The HSI also includes a number of configuration functions which are used to tailor IRAF to a particular host or environment. The HSI is fairly small compared to the remainder of the IRAF system. Although host-dependent, the code contained is relatively portable, especially across similar types of operating systems.

8.5.4 Applicability to the Concept Executive

The IRAF system as it currently exists is not directly applicable to the Concept Executive, as it is not an executive system as defined by this document. However, if the Concept Executive was being developed from scratch, IRAF could be used as the basis upon which the system could be built. This is much the same as if the system was built on top of UNIX, using standard system utilities and functions. While IRAF provides some advantages not found in UNIX, it does not provide much of the functionality required of a workstation executive. Namely, complete Local Area Network support, configuration management, distributed process control, and other functions.

While not directly applicable to the Concept Executive, the IRAF system nonetheless provides concepts and insight into solutions to problems. For example, a concern with the Concept Executive is that it is difficult to justify giving users access to the UNIX shell during operations. As done in IRAF, a special purpose shell could be developed and used for this purpose. This would allow any special security requirements to be handled directly by the normal user interface.

Another concern of the Concept Executive is how to insure portability of all system and user application software across different vendor platforms. At the time IRAF was developed, the concept of standard UNIX (SVID or POSIX), X Windows, GKS, and other standards were either not yet conceived or only in early stages of development. For a system such as IRAF, portability is a primary requirement, as it is needed to run on a variety of different systems at different sites. At the current time, standards have matured to the point where it is possible to write portable code. Using standard languages, standard UNIX (SVID and later POSIX), X Windows, GKS, and other standards will allow portable software to be developed.

In any large system, especially a critical system depending heavily on real-time performance, there will be host-dependent implementations in order to achieve optimum response. A lesson can be learned from IRAF in that such required functions should be front-ended such that the host-dependent details are shielded from the application (system or user). In this way, only a small portion of code must be modified in order to port to a new host.

Note that the use of a VOS is not really that unique. If you think about any application in an typical spacecraft support environment (such as the real-time data manager), you see an application built on top of host operating system functions. This application in turn provides programmatic (C callable functions) and interactive means whereby its functions can be selected and used. In this way, it is a VOS itself, albeit a very small and specific one. It is also one in which the interfaces to other such applications are cumbersome (such as the way data is provided for application computation or display).

In the same manner, most environments provide a VOS of sorts, however it is in the form of separate applications which are not well-integrated. Each of these applications provides its own language (set of commands) and/or an interactive command builder. A more suitable approach would be to evaluate the requirements of the environment's users and build a VOS which provides all the functionality. Next provide a complete language which allows users to exercise the functions on this VOS. The advantages of this approach are:

- Portability - the VOS functions are fully host-independent. All details are hidden beneath the VOS. In actuality, the VOS itself should be almost entirely host-independent.
- Extensibility - It will be simple to add a new function or set of function to the VOS.
- User-friendly - The language will be simple to use and will be consistent for all applications. It will also be nice to provide a means whereby a program is built interactively.
- Integrated - The entire editing process will be in one environment with all functions and commands available. The user will not have to move from one editor to another or result to clumsy interfaces to bring all data together.

Such an environment could be implemented at a level above the Concept Executive, thus providing a complete system for end user use.

8.6 TOAST System

The Transportable Operations Applications Software Task (TOAST) system was originally conceived by the Flight Dynamics Officers (FDO) at NASA/Johnson Space Center. The system was developed to provide flight planning, flight training, flight analysis, and flight support in the Mission Control Center (MCC).

TOAST is a software system which can execute under the Workstation Executive (WEX) environment on the Trajectory Mission Operations Directorate (MOD), Instrument Pointing System (IPS), Tactical Air Navigation (TACAN) Subsystem (MITS) Local Area Network (LAN) workstation complex in the MCC. The TOAST system can also be run in stand-alone mode to support preflight planning and training. It was originally developed on HP-9000 computers. TOAST supports the following trajectory application areas:

- Flight Dynamics Officer functions.
- Ground Navigation (NAV) functions.
- Onboard Navigation (ONAV) functions.
- Ascent Section functions.

The goals of the TOAST system are to provide the users with the capabilities required to support a mission in preflight, real time, and postflight modes. The system must support multiple users simultaneously on one or more machines connected by one or more LANs. The system must support dual real-time operations (defined as two separate flight and cycle combinations supported simultaneously with independent data bases).

The TOAST system provides an integrated set of verified applications which are designed to interface with the MCC data flow available on the LAN. The TOAST system supplies an executive structure and supported applications for a number of low-speed and high-speed trajectory functions. The executive structure contains the system's controlling logic along with certain standard services. It includes the following features:

- Control of user sign-on and database selection.
- A menu tree for selection of the desired applications and displays.
- Application dispatching, resource loading control, and result storage.
- Logging of user input commands sufficient to reconstruct configuration and computation histories.
- A checkpoint capability.

8.6.1 Contact Point

The TOAST system was designed by the Orbit Design Section of the Flight Design and Dynamics Division at NASA/Johnson Space Center. To receive more information on this system contact:

Orbit Design Section

NASA/Johnson Space Center

Houston, TX 77058

8.6.2 Review Process

The review process for the TOAST system involved examination of the following documents:

- TOAST Requirements Document, May 1987.
- TOAST Executive Program Guide, May 1988.
- Implementation Design for TOAST, June 1989.
- TOAST Volume I Executive, Draft, November 1989.

The documents reviewed provided an overview of the TOAST system and a discussion of the software design.

8.6.3 System Description

The primary purpose of a system such as TOAST which interfaces with the MITS LAN is for real-time command and data exchange with the Mission Operations Computer (MOC) and the Near Real-Time (NRT) data system. The MITS workstation provides supplemental computation and display capabilities as well as data reformatting and logging to disk or tape. The TOAST system provides the interface for these capabilities. The Trajectory MITS workstations are connected to the MITS LAN in order to send commands to and receive data and messages from both the MOC and the NRT System.

The TOAST system can be operated in a stand-alone mode if the MOC is not supporting the MITS LAN. In stand-alone mode, users can perform significant off-line analyses, such as planning for real time support or training exercises. For example, the FDO simulation (SIM) support requires generating significant amounts of trajectory profile data (state vectors, maneuver targets, etc) based on the mission plan. The TOAST system uses a unified approach to provide these off-line tools in a manner consistent with real-time application, thus simplifying operator training requirements.

The TOAST system has an open design allowing for additional applications to be added as needed. The system adheres to the following guidelines:

- The user interface is designed to be convenient, rapid, and user friendly. Nearly all TOAST commands are implemented via menu structure and function keys. Exceptions

are clearly noted. A shorthand command input capability is also available. Menus have logical defaults. Cursor control is by arrow keys.

- Application formulations and capabilities default to MOC requirements. However, additional options are available in many cases.
- LAN interfaces are provided via approved interface programs. The workstation software design does not preclude sending any valid Manual Entry Device (MED) commands to the MOC from the workstation. NRT commands are also supported.
- Transportability of code is highly desirable. All programming is done in FORTRAN or C and all hardware and system dependent code is modularized for containment and is documented.
- The system adheres to all security requirements for NASA/JSC. This includes requirements for converting software from the highly classified red machine to the less secure black machine.
- Initialization files are supported. The user may create unique initialization files for analysis as well as mission support. Each of the Dual Operations sessions permits the initial user to select the appropriate initialization file from among those available. Logging on by flight ID alone forces the selection of the approved flight initialization file for that flight.
- TOAST supports multiple terminals having displays controlled from one keyboard. The user is also protected from undesired output to his terminal. Conversely, any user is able to view data from any Session Data Area (SDA), but only the users of a specified SDA may write to that SDA.

There are thirteen functional areas in the TOAST system. They include:

- Password Program.
- Sign-On and Sign-Off.
- Application Selection Program.
- Session State.
- Command Methods.
- Job Execution.
- Database Manipulation.
- Display.
- Clocks.
- Discrete Digital Driver (DDD).
- Color Scheme.
- Monitoring and Logging.
- Common Library.

8.6.3.1 Password Program

The TOAST Password Program requires the user to enter a UNIX login ID and associated password. Once the login and password match, TOAST is executed. TOAST uses the UNIX login ID

of each user to determine application and flight access. WEX requires all users to login by flight control position and does not require unique user identification. The TOAST Password Program was created to replace the UNIX login removed by WEX. This allows TOAST to restrict user access to the system.

The user must enter a valid UNIX login ID and associated password. The login IDs and passwords are defined in the UNIX configuration file "/etc/passwd." The user is given only three chances to enter a correct login and password. Once the user has entered a correct login and password, TOAST is executed.

8.6.3.2 Sign-On and Sign-Off

The TOAST Sign-On Menu provides controlled access to the TOAST system, performs system status update, Session Data Area (SDA) initialization, and takes the user to the TOAST Application Selection Program. The user executes TOAST simply by typing "toast<CR>." Before the Sign-On Menu displays the menu for user inputs, it checks whether or not the user has access to TOAST. If the user is not authorized for access to TOAST, the sign-on attempt is logged to a file, the user is notified of the error, and the TOAST process is terminated. Otherwise, TOAST displays the program identification - name, version, and date - and brings up the Sign-On menu. The user inputs the position ID, flight ID, cycle ID, session ID, and the SDA initialization parameters. Each position ID within TOAST determines the menu structure for the user that is used by the Application Selection Program. The flight, cycle, and session IDs determine the SDA and the initialization data parameters define the source of the SDA data initialization.

Positions must be installed into TOAST by the TOAST administrator since the menu structure and applications are associated with a particular position. TOAST access is controlled by a configuration managed access file. This file defines the access to position and flight for each TOAST user. If the file does not exist, all UNIX users have unlimited access to TOAST. If the file does exist, only those users listed in the file have access to TOAST and the users only have the access defined in the access file. The file contains the following information for each user:

- User ID - UNIX login ID for each user allowed access to TOAST.
- Position ID - valid position IDs for the user; if no position IDs are given, the user has access to all installed positions.
- Flight IDs - associated with position IDs; indicate the flights which may be requested with each position ID; if no flight IDs are given, the user may request any flight.

When an SDA is created, the state of the SDA is defined as "private." This means that no other user may gain access to the SDA, regardless of the permissions defined in the access file. If the user wants the same position, flight, cycle, they must specify a different session. A TOAST menu exists that allows a user in an SDA to change the state of the SDA between "private" and "public." When an SDA is "public", other users may request access to that SDA and gain access.

The Sign-On Menu will also provide the user with several help keys that display valid parameters for the input fields. The help keys available are:

- Active Sessions - displays the sessions that are currently active.
- Access - displays the position and flight access for the user.
- Save Areas - displays either the users who have save areas, the save areas for a specified user, or the data within a given save area for a given user.

- Baseline Data - displays the names of the flight/cycle combinations that have baseline data available.

Once the user has completed the sign-on menu and selected the [EXECUTE] function key and TOAST has verified the inputs, the SDA is initialized. After the SDA is initialized, the Application Selection Processor is started using the menu structure defined by the position ID. An [EXIT] function key is displayed on the user's main menu and the user's session will continue until this option is selected.

When the user has selected to exit TOAST, the Sign-Off menu is displayed. The TOAST Sign-Off menu provides the user with the opportunity to save selected data tables from the SDA before signing off TOAST. In addition, it performs the inverse functions of the TOAST Sign-On menu. Where the Sign-On menu performs set-up functions, the Sign-Off menu performs certain wrap-up functions. The wrap-up functions performed are:

- Clear the position from the SDA - as each position terminates, the SDA files unique to that position are removed. When the last user for a position exits, the position is cleared from the SDA.
- Delete the SDA - if this user is the last user on the SDA, the SDA is deleted from the system.
- Update the TOAST status files - remove the user from the list of users on the SDA. If the termination of the user also terminates the SDA, remove the SDA from the list of active sessions.

During sign-off, TOAST will stop all executive functions associated with a user's session, delete all executive tables, table entries, and files associated with the user, and remove all references to the user from the executive bookkeeping. If the user has no batch jobs running and if no other user is using the same data area, the user's data area is cleared. Finally, the log file is updated to reflect the sign-off, and the user is dropped from TOAST.

From the Sign-Off menu, the user is given the option of saving the session data to permanent media. The Sign-Off menu displays the contents of the user's SDA and allows the user to save specified data to a Save Area (SA). The user may save data to an SA as many times as desired. The data save component is accomplished by using the Database Manipulation Program which is discussed in a later section. When the user has finished saving data (or chosen not to), he may either press the [EXIT MENU] key to exit TOAST or press the [TOAST] key to return to TOAST.

8.6.3.3 Application Selection Program

The Application Selection Program (ASP) provides the user with a 3-tiered menu structure that allows the user to select applications to execute. This program is started by the Sign-On Menu program after the user's inputs have been verified and the SDA initialized.

The main menu, identified by a user's position, is initially displayed after successful sign-on to TOAST. Each option on the main menu has an associated sub-menu that is displayed when the user selects that option. Each sub-menu option, in turn, has an associated user program. When an option is chosen from the sub-menu, the program associated with that option is invoked. The programs selected from the sub-menu are the position-specific menu and display programs.

The menus contained within the ASP are choice menus and therefore have no user inputs. These menus include:

- Main Menu - contains a list of application categories from which to choose. When the user chooses an option, the corresponding sub-menu will be displayed. If the [EXIT] option is selected, the Sign-Off menu program is executed.
- Sub-menu - contains a list of user-supplied applications to invoke. When the user makes a selection, the application is executed. If there is a problem executing the selection, an error message is displayed and the current option is highlighted. Once the requested application has completed, the sub-menu is redisplayed with the last chosen option highlighted. Pressing the [EXIT] option causes the main menu to be redisplayed with the last chosen option highlighted.

8.6.3.4 Session State

The TOAST Session State application allows the user to toggle the state of a TOAST session between public and private. When a session is public, any number of users may sign on to that session. When a session is private, only those users logged into the TOAST session when the state was set to private can sign on to the session. If another user attempts to sign on to a private session, a message will appear on the Sign-On menu stating that the session is private and the user will not be allowed into the session. When a session is created, its default state is private. When the last user has signed off of the session and batch jobs are left running, the session will remain in the state it was in when the last user signed off. The TOAST Session State application displays the flight, cycle, and session that were entered on the Sign-On menu along with the current state of the session and a scrolling list containing the active users in the TOAST session along with their position IDs and terminal IDs. The user changes the state of the session by depressing a function key.

8.6.3.5 Command Methods

TOAST has three different means of entering commands - menus, function keys, and the command line. The menu structure is the primary command method for TOAST. All inputs may be entered through the menus. The function keys, when not application specific, are used for menu control. The command line serves as a shorthand alternative to the menus. There are also some system functions that may be accessed only through the command line. Each of the command methods occupies a specific part of the CRT display.

There are three levels of menus. The first level is the main menu whose choices represent a selection of application categories. For each Level 1 menu option, there is a Level 2 menu containing all the applications available in that category. The Level 3 menus are the input menus for the individual application programs. Each time the user calls up an input menu, the default values are displayed. The first time a menu is used, the TOAST-defined defaults are used. Otherwise, the default values are the most recently saved values for that menu. Values for a menu can be saved by the user by selecting the [EXECUTE] or [SAVE MENU] function key.

The function keys are used for menu control. A function key is pressed to indicate that the user has finished with a menu, or a portion of a context-dependent menu, and is ready to take some action. Each application may also use the function keys.

The TOAST system uses eight standard and "n" non-standard function keys. The functions of the standard keys can only be accessed through the function keys. The non-standard keys are assigned escape code sequences. The functions of these keys may be accessed by either the function key itself or the escape sequence assigned to the key. The executive function keys may be either standard or non-standard. The use of non-standard function keys is limited to "extra" keys on the key-

board. The label on non-standard function keys should not indicate a function not performed by that key. Application specific function keys will always be standard keys.

Each menu level has its own set of executive function keys and each level's keys are a superset of the previous level. If either the executive or an application requires more than four function keys on a Level 3 menu, f9 can be used to toggle the functions keys, and labels, to an alternate set. The function keys identified for each level are:

- Menu/Command (Standard) - toggle between menu and command mode.
- Save Menu (Standard) - saves the current menu inputs to a menu file.
- Execute (Standard) - saves the current menu inputs to a menu file and actually causes the execution of the application associated with the menu.
- Exit Menu (Standard) - exits the current menu and returns to the previous level's menu.
- Clear Menu (Non-Standard) - replaces the current menu defaults with the initial defaults.
- Clear Field (Non-Standard) - erases the value of the field at the current cursor location on the menu.
- Print Screen (Non-Standard) - prints the current contents of the screen to the terminal printer.
- Alternate Set (Non-Standard) - causes an alternate set of function key definitions to be used.

The command line is used for system functions and shorthand commands. The system functions are not available via the menu structure although some of them are available via function keys. The shorthand commands are all available via the menu structure.

The system functions are menu aborts available at the Level 3 menus. They provide a means of backing out of a menu without changing the state of the menu defaults. They are available when the user has toggled to command mode. The available system function commands include:

- (main) - takes the user out of the current menu and back to the main menu.
- (exit) - takes the user out of the current menu and back to the previous level 2 menu.
- (clear) - replaces the current menus defaults with the initial defaults.

When in the command mode, shorthand commands may be used instead of the menus to control applications. When using the menus, the user must follow the proper menu path to accomplish a task. When using the shorthand commands, the user may execute any command from anywhere. There is a one-to-one correspondence between the menu options and the shorthand commands.

In general, the shorthand commands do the following:

- Go to a Level 1, 2, or 3 menu.
- Execute an application.
- Bring up a display.

8.6.3.6 Job Execution

An application is defined as any job run as a result of a Level 3 menu request, or the equivalent shorthand request. There are three types of applications - processors, displays, and interactive displays. The definition of each is given below:

- Processor - an application that processes data but which does not generate any screen output.
- Display - an application which reads data from a table and, with little or no computations, prints the table on the screen.
- Interactive display - a display that allows the user to change the data displayed after it has been printed to the screen or that allows the user to manipulate a data table that is being displayed.

An application is executed in one of two job categories - demand or non-demand. Non-demand jobs are those jobs that are placed in a common job queue, first in first out (FIFO) style. The jobs in this queue are sent to the CPU for execution using a simple scheduling algorithm. This prevents the applications from overloading the system. Demand jobs, on the other hand, are jobs that need to be executed immediately. When an application is run as a demand job, it is not placed in the job queue and the user is not allowed to proceed to another application until the demand job is finished. Demand jobs include displays, interactive displays, the [EXIT] option on the main menu, status inquiries, and clock functions. All other applications are run as non-demand jobs.

Non-demand jobs are further categorized into queued and batch so that there are essentially three categories:

- Demand job - a job that runs immediately without going through the job queue.
- Non-demand queued job - a job that goes through the job queue and terminates if the user signs-off before the job is finished.
- Non-demand batch job - same as queued job except that it runs to completion when a user signs-off before the job is finished.

8.6.3.7 Database Manipulation Menu

Data in TOAST is organized so that there are constants, flight data, cycle data, and session data. This allows dual flight operations. Actually, it allows as many operations as the disk space allows. Users may be on the same flight/cycle but different sessions, or they may be on the same session. The constants, flight data, and cycle data are maintained by the TOAST system manager. Users are allowed to copy this data into their Session Data Area (SDA) but cannot write to the system database.

TOAST maintains its own database of information needed to operate and coordinate the user's activities. Items like users signed-on, display locations, sessions, job status, and module status are recorded. Although the users will use this data, they will have no direct access to it and cannot write to these areas.

Each user signs-on to one pre-defined session. The flight/cycle and data IDs given by the user determine the user's SDA. Each session has an associated SDA and save area (SA). The user maintains these areas through TOAST. A user may read from another SDA or SA, but cannot write to them. A user also has access to his own SA external to TOAST. The super-user has access to all SDAs and SAs internal and external to TOAST.

The SDA of a user is defined by flight/cycle ID and data ID. If more than one user gives the same IDs, then these users have both read and write access to the same SDA. TOAST, itself, insures data interface table (DIT) integrity, but the users of an SDA must coordinate to insure the integrity of the entire SDA. Each user is given the option of saving all current DITs to an area outside of

the TOAST domain which the user has configuration control over. This can be done anytime during a session or at sign-off. The SDA will only be cleared by TOAST when the last user has signed-off of a session and there are no batch jobs running.

Unless a session was left with batch jobs running, when a user signs-on to TOAST, the SDA is empty except for flight/cycle data. The user may then load the SDA or proceed to execute applications to build the required session specific data. Sources of SDA loads include flight constants, flight data, cycle data, other SDAs, SAs, and removable media. An SDA may be saved/loaded in its entirety or by individual DITs.

Each application uses specific DITs during execution. Since TOAST allows many applications to run in parallel, TOAST provides a means to protect a table from simultaneous access by more than one application. The application locks the table before using it and does not unlock it until the application is completely done with the table. This insures that an application accesses a table whose data is consistent. Coordination of table to table consistency is left to the users.

The TOAST Database Manipulation menu provides the user with the opportunity to save selected tables from the SDA to a specified SA, to load selected tables from a specified SA to the SDA or to view the contents of a specified SA. When the database menu is initially displayed, information about the user's SDA and the contents of the SDA are displayed. The user may request to load data into the SDA, save data into an SA, or view the contents of an SA. If this application is being used for the Sign-Off menu, the user may not load data into the SDA. To load from an SA to the SDA, the user must specify the name of the user whose SA is being used, the name of the SA, and the list of tables to load. To save data to an SA, the user must specify the name of the SA and the list of tables to save. When saving to an SA, the user may only save to his own SA.

The Database Manipulation Menu contains three sections:

- Header Section - contains information about the tables being displayed in the table section of the menu. It contains:
  - Position ID of the current user.
  - Flight and cycle IDs of the tables displayed.
  - Session name of the tables.
  - The name of the current user, if an SDA is displayed or the name of the owner, if an SA is displayed.
  - The name of the SA, if the tables are from an SA.
- Table Section - displays the tables. If there are more tables than can be displayed, the user may scroll through the tables. Each line in the section contains:
  - The index of the table.
  - Current status of the table: save (S), existence (D), and classification (C).
  - The table descriptor, not the filename.
- Data Entry section - allows the user to:
  - Display the tables in the current SDA.
  - Display the tables in the specified SA.
  - Save data from the current SDA to the specified SA.
  - Load data from the specified SA to the current SDA.

8.6.3.8 Display

The TOAST system provides a coordinated mechanism for displaying the results of applications and the contents of the database. Each user can display the results of his processing on his terminal, independent of all other users. The display to be viewed is selectable from a set of pre-defined formats. These formats mimic analogous MCC displays, to the extent possible, in appearance, parameter definitions, etc. Display formats for which no MCC analogs exist are also supported.

After a particular application finishes processing, the output parameters required to generate an associated display are stored in one or more DITs. A warning message is displayed on the applicable terminal's screen and sent to the on-line printer whenever a DIT is overwritten by reading from a file or floppy disk. This message is the user's warning that display data which depends on the overwritten table may be invalid.

Each user can request a display at his terminal based on the data contained in any DIT currently existing within TOAST. The resulting display contains a message identifying the SDA on which it is based. However, the application software can only write data into a user's own SDA.

The DITs in each SDA provide sufficient data to the display processing logic to generate any requested display. This data may include all of the display parameters directly, or it may only include some of the display parameters with the remainder of the parameters derivable from the DIT data, via computations to be performed within the display generation logic itself.

An error message is output to the requesting terminal if sufficient data cannot be retrieved from the specified SDA to satisfy a specific display generation request. Unless specifically allowed by the particular display logic, this error also terminates the display request processing - i.e., no partially complete display outputs are allowed.

Display generation requests default to the requesting user's SDA as the source for display data. The user can override the SDA source, but it must be specifically overridden for each request made based on another SDA.

A display DIT consistency monitoring task checks the compatibility between the displays currently selected at each of the terminals and the DIT data on which those displays are based. If an update is made to the DIT data that is driving any currently selected display, the display generation logic for that terminal is automatically queued to regenerate the display based on the updated DIT.

No default displays result from non-demand applications processing. All displays must be specifically requested by the operator.

Hardcopies made at the terminal include a terminal identifier, the current actual Greenwich Mean Time (GMT), and the flight/cycle identifier on the resulting hardcopy. Displays may be requested from the appropriate Level 2 menu, from the global display request menu or from the command line. Display request operations required by the user are minimized via the command line.

8.6.3.9 Clocks

The TOAST system provides for the display of several time of day clocks and user timers. The requirements for the clock display logic are terminal specific.

The TOAST clock area occupies the top line of the CRT and will allow the display of four clocks. The clock area is "windowed" such that it may be seen or placed in the background (invisible) at the user's discretion. The window defaults to ON (clocks visible) at a user's sign-on.

The user may have eight clocks running at any time, but only four clocks may be displayed. Countdown clocks are indicated by a minus sign preceding the time field while countup clocks use a plus sign. Two clocks are automatically started after TOAST sign-on. They are:

- SGMT - simulated GMT; default = GMT.
- MET - mission elapsed time; default = SGMT reference.

Clocks continue to operate regardless of other tasks being performed. If data does not exist to drive a requested clock, the digital time field on the display will be filled with asterisks instead of the time.

8.6.3.10 Discrete Digital Drivers

The TOAST system provides for the display of several discrete digital driver (DDD) event lights on the terminal screen. The DDDs on each terminal are driven by the data from the user's SDA and the format is defined by the user's session ID.

The TOAST DDD area contains 22 fields on lines 22-23 of the CRT. The user may specify the label to be printed in each DDD field. The DDD area is "windowed" such that it may be seen or placed in the background (invisible) at the user's discretion. The applications manipulate the DDDs and may turn on/off any number of them at one time.

The DDD states are updated once every second. The DDDs are not buffered so that state changes received at a rate of more than once per second will be lost except for the last state before each update.

8.6.3.11 Color Scheme

The TOAST menus and displays follow a standard color scheme. Display colors are dependent on the data to be displayed. No data is dependent on the color for meaning. Color is used as a secondary means of enhancing the display. Inverse video, blinking, and other types of non-color dependent modes of display are used to convey special meanings. This approach insures that the system is usable on monochrome displays and allows users who cannot discriminate between colors to still use the system.

8.6.3.12 Monitoring and Logging

The TOAST system maintains status information concerning most actions that take place (e.g., users logging on and off, application being executed). In addition, certain messages or data are logged for future reference. TOAST is a disk-oriented system. The status and logging information primarily resides in disk files. Displays are available to view this information on a terminal. The displayed information can be hardcopied at the terminal printer or the system printer. A user may print/view all or part of the log file. The user can print/view a part of the log file by user, session, or date, or a combination of these selections. The characteristics of the TOAST logging system are:

- User sessions are logged and a complete history is maintained.
- All warning and error messages are logged.

TOAST traps all system errors that are capable of being trapped, logs the information, and then performs necessary housekeeping before allowing the faulty process to die.

The user can query the system status for such items as:

- TOAST system status.

- List of current users along with their flight/cycle and SDA.
- Job history for the requesting user.
- Current job status TOAST-wide.
- Flight/cycle sets available.
- Clock status for the requesting user.
- Save Area (SA) or Session Data Area (SDA)contents.

8.6.3.13 Common Libraries

There are eight executive libraries. All of them are used by the Executive. Some are used by applications, both directly and indirectly. These libraries include:

- Util - contains routines that are independent of TOAST; That means they can be used outside of TOAST. These functions are small, general routines.
- Exec - contains the lowest level of TOAST routines; They require Lib_Util. These routines can only be used in TOAST as they use the database structure, Executive data files, or semaphores.
- Locks - contains the lock routines and associated support routines.
- Logging - contains logging and error handling routines; They are put in the same routine because the logging function is used in error handling. There are two error handling routines - one for Executive errors and one for application errors.
- Database - used by the Sign-Off menu (main program), the Executive application and the Database Manipulation Program.
- Submit - contains the job submittal routine; It has its own library to allow for the expansion of capabilities.
- Interface - is the FORTRAN to C interface library; Not every routine has an interface, not even every routine used by FORTRAN. Only the routines that need strings have an interface.
- Test - contains the generic routines for testing the libraries; This is really a library of a main routine and supporting subroutines. Each library has another specific subroutine for testing its routines. This program is data driven so each library (module) has a number of associated data files.

8.6.4 Applicability to the Concept Executive

The TOAST system is applicable to the Concept Executive in that it is a high-level interface between the user and the underlying system. Overall, the requirements for TOAST are similar to those of the Concept Executive. The major difference between the requirements of TOAST and the requirements of a Concept Executive lie in their scope. The TOAST system's scope is more limited than the Concept Executive in that it was designed to manage the display and manipulation of only the trajectory data in the MCC. The Concept Executive needs to manage all data and provide a secure and protected environment for all users of the system. TOAST also provides the tools for displaying and viewing information, while the Concept Executive should provide the interfaces to be used by those tools.

A primary requirement for TOAST was to make the interface consistent across all applications. This requirement is also necessary for the Concept Executive. Any system that has to interact with a user to perform its function must provide a consistent interface to the users of the system. TOAST maintains a consistent interface by providing standard function keys which cause the same result whenever the key is selected. Another requirement for consistency lies in the methods used to interface with the system commands. TOAST provides the user with a limited number of new commands which may be initiated through the command line as well as through function keys and shorthand commands (escape sequences). The use of shorthand commands, such as those used in TOAST, is an admirable quality for any command interface and should be a consideration in the requirements for the Concept Executive.

The message logging system provided by TOAST can be seen as a beginning to the security considerations necessary for the Concept Executive. The TOAST logging capability could be used to audit a user's activity and the activity on a particular system entity.

TOAST provides data protection to the user extensively through the standard UNIX owner and group permissions. This method of user data protection could also be used by the Concept Executive. TOAST assigns different data areas depending on the inputs entered by the user at sign-on. If a user attempts to sign-on to a session with the same data area as another user, TOAST will either not allow the user to sign-on (if the data area is "private") or will inform the user of the other users signed-on to the same data area (if the data area is "public"). This method potentially means there could be high amounts of duplicated data between different data areas. If each data area requires large amounts of space then it would be inefficient to implement this method of data protection in the Concept Executive. A modification of this method, using a common area for duplicated data could be a potential resolution to this problem. In this case, the common area for all data areas would need to be controlled by a locking mechanism similar to the one used for a "public" data area.

One of the reasons for TOAST's implementation of data areas, is that it allows for separate operations in a single flight and it allows multiple users simultaneous access to common data. This is also a requirement for the Concept Executive, however it appears that the TOAST implementation would need to be enhanced in order to meet all of the requirements of data protection in the Concept Executive.

Since TOAST runs under the current workstation executive in the MCC, it does not provide any special configuration management, network, or LAN data acquisition functions.

8.7 SFOC System

The Space Flight Operations Center (SFOC) is a system in use and under development at the Jet Propulsion Laboratory (JPL). One of the primary responsibilities of the JPL is to provide ground and scientific processing support for long range planetary exploration spacecraft, such as Voyager, Magellan, and Galileo. The JPL has been using the Mission Control and Computing Center (MCCC) to support the different spacecraft under its responsibility. The JPL is in the process of replacing this dated system with the SFOC system. The SFOC system is a collection of hardware and software which supports multiple missions and provides the baseline for support of current and future spacecraft.

When the SFOC project was originally conceived, the scope of the project was to provide full capability for the new Magellan and Mars Observer missions, and converting the existing Voyager and Galileo missions to the SFOC environment. All work was to be completed by the end of 1989.

Three factors contributed to a change in this plan. First, the Mars Observer launch date was changed from 1990 to 1992. Second, funds were not available to develop all SFOC systems during the 1986, 1987, and 1988 time frame. Only those subsystems required to support Magellan launch were implemented. Finally, funds were not available nor was the timing appropriate for conversion of either Voyager or Galileo.

A new plan approved by NASA extends the SFOC project to cover the original scope of work. This work will be completed as soon as possible in order to completely replace the MCCC and therefore eliminate the requirement to support two separate control environments. The new project plan is called the "extended" project plan.

The SFOC as currently implemented consists of a baseline set of subsystems required to support the Magellan launch in May 1989. This review will concentrate on this implementation, as this baseline represents the current system and includes the "core" subsystems which most clearly relate to the Concept Executive.

8.7.1 Contact Point

Space Flight Operations Center

NASA/Jet Propulsion Laboratory

8.7.2 Review Process

The review process for the SFOC system was complicated by the changes in project scope and the related efforts to bring all documentation up to date. Most of the documents received had not been updated for the changes in project scope.

The following set of documents represents the SFOC design plan. The design plan provides the high-level system guidelines and the level 1, 2, and 3 functional requirements. These documents were generated in January and February of 1986 and had not been updated to reflect the changes in project scope. These documents were scanned nonetheless to obtain an overview of the complete SFOC system. The documents reviewed include the following:

- Space Flight Operations Center Design Book - Introduction.
- Space Flight Operations Center Design Book - Guidelines and Constraints.
- Space Flight Operations Center Design Book - JPL Control Center Functional Responsibilities.
- Space Flight Operations Center Design Book - SFOC Functional Requirements.
- Space Flight Operations Center Design Book - SFOC Data Processing Architecture.
- Space Flight Operations Center Design Book - Flight Operations Subsystems Requirements.
- Space Flight Operations Center Design Book - Planning and Analysis Subsystems Requirements.
- Space Flight Operations Center Design Book - SFOC Data System Requirements.
- Space Flight Operations Center Design Book - SFOC Operations Organization.

The next set of documents reviewed was the updated SFOC project plan. These documents provided a high-level description of the functional makeup of the SFOC system. There was a set of "plan" level documents which completely describe the SFOC extended project plan (primarily in

terms of procedures and responsibilities). It appears that all other documentation is being updated to reflect this plan, but as of the date of this review, this process was not yet complete. The "plan" documents reviewed include the following:

- Space Flight Operations Center Project Plan.
- Space Flight Operations Center Documentation Plan.

The final set of documents reviewed consisted of the level 4 functional requirements for several of the subsystems present in the baseline SFOC system. As indicated, only those subsystems applicable to executives were reviewed:

- Space Flight Operations Center Workstation Support Environment (WSE) Functional Requirements.
- Space Flight Operations Center Data Transport System (DTS) Functional Requirements.
- Space Flight Operations Center Common Data Access (CDA) Software Specifications Document.
- Space Flight Operations Center SFOC Monitor and Display (SMC) Functional Requirements.

Ideally, both the functional requirements and software specifications would have been reviewed for each subsystem. This however was impossible as only the listed documents were provided.

8.7.3 System Description

Development of the SFOC system is a multi-year effort which is intended to develop and bring into operation an upgraded mission control center. The SFOC system is intended to replace the existing MCCC system, which relies on outdated, high maintenance cost technology. The SFOC system will use up to date computing technology to satisfy the requirements of a number of flight projects. The primary goals of the SFOC system are as follows:

- Provide an innovative, lower cost approach to ground support of planetary exploration spacecraft.
- Eliminate the requirement for unique ground support hardware and software by sharing common hardware and baseline software.

The SFOC system achieves these goals by defining a multimission baseline which minimizes the amount of flight-specific development and resources. The SFOC system is intended to support multiple flight projects, which are defined as the unique characteristics of a given spacecraft flight, including hardware, software, data, and support personnel. By providing a flexible baseline system, the SFOC allows cost effective support of new flight projects. For each new flight project, the baseline system is adapted to support the unique requirements. The procedure for baseline adaptation is as follows:

- Provide new parameters for insertion into pre-defined tables.
- Create new display formats.
- Add additional hardware as necessary to support the processing requirements of the flight project.
- Add new data to the database.
- As a last resort, change or add subsystem software.

8.7.3.1 SFOC Guidelines and Constraints

The SFOC system functional requirements and design are affected by a set of guidelines and constraints. These guidelines and constraints are presented to provide an overview of the specific goals of the SFOC system:

- System guidelines and constraints:
  - Provide hardware and software expandability in order to meet the requirements of new flight projects.
  - Provide automation of labor intensive processes.
  - Allow introduction of new hardware and software technology.
  - Minimize the number of different types of data processing equipment required for environment support.
- Fundamental design guidelines and constraints:
  - Use a Local Area Network (LAN) as a means for communicating among system elements.
  - Individual SFOC functions will access an integrated database via a Data Base Management System (DBMS).
  - SFOC functions will be implemented on appropriately sized processors. Most functionality will be on workstations.
  - Accommodate local and remote users (scientists at different physical locations).
  - Provide fault tolerance and safeguard critical mission data.
  - Use distributed computing utilizing local area network and workstation technologies.
- Design guidelines and constraints:
  - Use national and international standards for data exchange. Use JPL Information System Standards, including Standard Format Data Unit (SFDU) for data to be exchanged between the SFOC and other systems.
  - Use layered approach to networking as defined by the Open Systems Interconnection (OSI) standard.
  - Provide secure access to system processing capabilities and to data stored by the SFOC.
  - Be designed in a manner which allows testability.
  - Maximize use of COTS technology for hardware and software.
  - Maximize use of vendor-independent solutions.
  - Incorporate use of existing shared computing resources.
- SFOC user guidelines and constraints:
  - Extensive use of software standards including UNIX, C FORTRAN, X Windows, TCP/IP and SFDU.
  - Use Ethernet for communications with goal of migration to Fiber Distributed Data Interface (FDDI) when feasible.
  - Use standards for telemetry data packetization and channel coding, telecommand data, and time-code formats.
  - Use data management standards.
  - Provide a high-level user command mechanism such as System Test and Operations Language (STOL) or Transportable Applications Environment (TAE).
  - Devote special attention to user interfaces.

8.7.3.2 SFOC Baseline

The starting point for the extended SFOC project plan is the baseline system developed for support of the Magellan launch. The SFOC baseline consists of 3 core and 7 application software subsystems. The core subsystems, which provide the lowest level of common support, include the following:

- Workstation Support Environment Subsystem (WSE) - provides the data processing, user interface, and display environment in which most SFOC subsystems and applications are executed.
- Data Transport Subsystem (DTS) - provides data communications and transport services for SFOC subsystems and computing nodes.
- Common Data Access Subsystem (CDA) - provides access to data storage and retrieval using Data Base Management System (DBMS) and file management services. This subsystem provides centralized, implementation dependent data management services to other subsystems.

The 7 application subsystems, which are developed using functions of the core subsystems, include the following:

- SFOC Monitor and Control Subsystem (SMC) - monitors performance and provides a means of controlling the SFOC environment.
- Central Database Subsystem (CDB) - also known as the Mission Operations Database (MODB), stores, retrieves, catalogs, and archives SFOC data. All data is stored in a relational database.
- Ground Communications Facility (GCF) Interface Subsystem (GIF) - provides the interface to the Deep Space Network (DSN) and the Interim Simulation Subsystem. The GIF subsystem captures and routes incoming telemetry and outgoing command data. External data formats used by the DSN are reformatted into internal structures used by the SFOC system.
- Telemetry Input Subsystem (TIS) - provides input processing on telemetry frames and DSN monitor blocks. Telemetry processing includes frame synchronization, decoding for error correction, synchronous and asynchronous extraction, depacketization, decommutation, and channelization.
- Data Monitor and Display (DMD) - provides standard processing and display of telemetry and other channelized types of data. This includes real-time, near-real time, and archived data. Display types include plots, fixed and variable matrices, and lists.
- Digital Television Subsystem (DTV) - generates displays of telemetry and other data for distribution via the Closed-Circuit Television facility (CCTV).
- Test Workstation (TWS) - provides data stream analysis and troubleshooting capabilities to aid in problem identification and isolation.

The baseline system provides downlink support of low-rate data for the Magellan flight project. The current downlink data flow is illustrated in Figure 8-9.

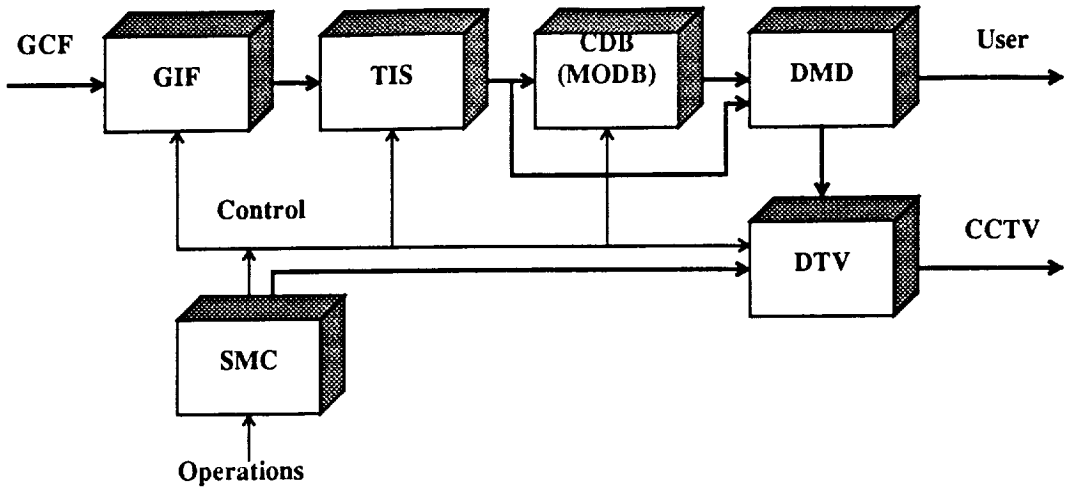


Figure 8-9 Baseline System Data Flow

The high-rate downlink (high speed telemetry) and uplink (command) functions required for the Magellan flight project will be provided in later deliveries.

8.7.3.3 Non-Baseline SFOC Subsystems

In addition to the subsystems which make up the SFOC baseline, a number of additional subsystems will be developed to provide complete support for Magellan and other flight projects. These subsystems are described as follows:

- Data Products Subsystem (DPS) - responsible for collecting engineering, ground monitor, and science telemetry data (as well as other ancillary data) for generation of specific data products for use by science investigators and analysts.
- Data Management Subsystem (DMS) - provides the data storage and retrieval capabilities for the SFOC. Data storage may be local to a node, centralized, or archived.
- External User Access Subsystem (EUA) - provides a means of access to the SFOC system for external users. This subsystem will provide access to scientists at physically remote locations.
- Sequence Subsystem (SEQ) - provides a means of supporting the uplink process. This subsystem will be used to translate desired spacecraft actions into the necessary commands.
- Command Subsystem (CMD) - provides a reliable, concurrent mechanism for commanding multiple spacecraft in the SFOC multimission environment.
- Navigation Support Subsystem (SNAV) - provides multimission support to the spacecraft navigation functions performed by the Flight Project Navigation team (FPNAV) for each flight project.
- Simulation Subsystem (SIM) - provides a means of simulating the Deep Space Network (DSN) functions to provide a source for testing of all SFOC subsystems.
- Multimission Image Processing Subsystem (MIPS) - provides a multimission facility for image processing support to flight projects.

- Engineering Analysis Subsystem (EAS) - provides support for engineering data analysis for multimission and project users.
- Telecommunications Analysis Subsystem (TAS) - provide tools and services that enable flight project users to predict telecommunications performance and to compare predicted performance with actual performance.
- Mission Analysis Subsystem (MAS) - provides tools and services for mission design and analysis.
- Science Analysis Subsystem (SAS) - provides the capability for evaluation of instrument health and status, first order analysis of scientific data, mission/sequence planning, and sequence verification prior to command uplink.
- Magellan High Rate Subsystem (MHR) - provides support for the Synthetic Aperture Radar (SAR) device which will be used to obtain detailed data for the surface of the planet Venus.

8.7.3.4 SFOC Baseline Subsystems Details

This section will describe in more detail several of the subsystems which make up the SFOC baseline system. It is beyond the scope of this document to describe each of the 10 subsystems in detail, therefore, only a selected subset of the subsystems will be reviewed. The subsystems reviewed include those representing the core functionality and one other system which is considered particularly applicable to the Concept Executive. The subsystems included are as follows:

- Workstation Support Environment Subsystem (WSE).
- Data Transport Subsystem (DTS).
- Common Data Access Subsystem (CDA).
- Data Monitor and Display (DMD).
- SFOC Monitor and Control Subsystem (SMC).

Although not a core subsystem, the SMC subsystem was reviewed due to its role in monitoring and controlling the operation and configuration of the SFOC environment.

8.7.3.4.1 Workstation Support Environment Subsystem (WSE)

The Workstation Support Environment Subsystem (WSE) will provide a uniform base environment to the users of all SFOC workstations. The WSE forms the basis for access and interaction with all workstations in the SFOC environment. The WSE defines the basic functional environment which allows execution of applications and on some workstations, application development.

To provide the uniform base environment, the WSE will provide the following functions at all workstations:

- Login verification.
- A common user interface to applications.
- A data retrieval utility (for non real-time data).
- lLmited file protection mechanisms to insure data integrity.
- A backup and archiving utility.
- A workstation to workstation file transfer mechanism.

- The ability to communicate with other SFOC users.

The WSE will also allow application development on certain workstations. This requires the following additional set of functions:

- A standard application development environment
- A graphics development environment (libraries)
- A window application development environment (libraries)
- Access to high resolution plotters and hard copy devices
- A window manager to create multiple virtual terminals

The WSE is divided into SFOC and vendor supplied software. The following two sections describe each.

8.7.3.4.1.1 SFOC Supplied Tools

This section describes applications, libraries, and tools provided by the SFOC system (as opposed to vendor supplied COTS functions).

The WSE will provide a user interface which will be used by all applications. The user interface will provide the following:

- A command interface, which allows execution of applications, creation and use of macros (command files), and control of the system. The requirements indicate a command interpreter with much the same functionality as the UNIX C shell.
- A menu interface, which uses menus to present lists of commands. When a menu item is selected, the corresponding command is executed.

The WSE will provide a number of stand-alone applications, which once run via the command or menu interface, may be used without command/menu interaction. The stand-alone applications include the following:

- A data retriever, which allows the user to make a database query and bring data from the central database into a local database where the data can be reviewed.
- A display template application, which allows definition of the type and layout of a display. This application allows users to define the format and data content of displays. Support is provided for list, matrix, alarm, message, tabular, and plot data types.
- A menu creation/maintenance utility, which allows users to define and modify the menus described above.
- A Standard Format Data Unit (SFDU) filter, which allows users to filter files of SFDUs based on keys in the data.

The WSE will provide several libraries of routines required on application development workstations. The libraries provided include the following:

- Data retrieval routines.
- SFDU filter and metering routines.
- Display template support routines.
- SFOC Monitor and Control Subsystem (SMC) interface routines.

The WSE will provide administrative applications which allow control of the workstation environment. This includes the following:

- User account maintenance and peripheral support.
- Anomaly reporting for violations and errors.
- Global environment editing.
- Security for files and other objects.
- Control of printers and other shared resources.
- Control of local devices.

8.7.3.4.1.2 Vendor Supplied Tools

The WSE specifies a number of vendor supplied tools which are required on all workstations. Most of these requirements are satisfied by functions provided by the standard UNIX operating system.

The WSE provides a number of tools which facilitate LAN or workstation communications. These include:

- User to user communication on local and remote workstations.
- Internal SFOC mail and bulletin boards.
- An interactive communication facility.

The WSE will provide a program development environment which supports several high level languages. The program development environment will provide:

- C and FORTRAN 77 compilers.
- C and FORTRAN 77 source level debuggers.
- A source code control system to allow software configuration management.
- Window and graphics development libraries with bindings available in C and FORTRAN.
- A screen oriented text editor.

The WSE will provide a number of miscellaneous tools useful for interactive and general workstation maintenance use. These include the following:

- Calculators and/or interpreted languages to allow basic mathematical calculations to be performed without developing an actual program.
- Text manipulation tools.
- Backup and archival tools.
- File transfer tools to allow movement of files from one workstation to another.
- Remote login to other workstations.
- File display utilities.

8.7.3.4.2 Data Transport Subsystem (DTS)

The Data Transport Subsystem (DTS) resides on each node and provides a data communications interface to other nodes in the SFOC environment. The DTS supports access to both internal and

external users. Internal users are those which are present on SFOC nodes. External users are those at remote sites which are accessing the SFOC via the External User Access Subsystem (EUA).

The DTS will provide all required hardware and software to allow communications between all nodes in the SFOC network. The DTS hardware will be based on COTS local area network (LAN) technology. The DTS software will be based on the Open Systems Interconnection (OSI) model, will support transparent routing, will support remote user access, and will be functionally identical on all nodes. It appears that the DTS will use TCP/IP and related protocols for network communications.

The DTS will provide a set of communication services required by other subsystems. The major services provided include the following:

- Remote login.
- File transfer.
- Common data representation.
- Process to process communications, including broadcast, point to point datagrams, virtual circuit, and request-response.
- SMC interface.

Each of these communications services will be described in more detail in the following sections.

8.7.3.4.2.1 Remote Login

The DTS will provide a remote login function which allows a user on the local node to establish a terminal-like session on a remote node. This requires that the requesting user have an account on the remote node. Once the session is established, it will appear that the user is directly connected to the remote node.

8.7.3.4.2.2 File Transfer

The DTS will provide a file transfer function which allows files to be exchanged between the local and remote nodes. The transfer process will honor all user and file permissions and will handle data differences between nodes.

8.7.3.4.2.3 Common Data Representation

The DTS will provide a common data representation which allows interchange of data between different machines. This function will translate local data into the standard format for subsequent transmission and will convert standard format to local format for incoming data. This format will be used for all incoming and outgoing communications. Libraries will be provided for format conversion for user applications.

8.7.3.4.2.4 Process to Process Communications

The DTS will provide an open-close-send-receive model of process to process communication (PPC). At open time, the user will be able to select any of the following four types of PPC services:

- Broadcast datagram - a non-guaranteed mechanism for sending a message to any number of other processes.
- Point to point datagram - a non-guaranteed mechanism for sending a message to any single process.

- Virtual circuit - a guaranteed mechanism for sending and receiving messages to a single process.
- Request-response - a guaranteed mechanism for sending a message and receiving a single response message (with a single process).

The DTS will provide a means of relating logical names to physical node addresses. These logical names uniquely define the nodes in the SFOC environment.

All PPC services will be message oriented, in which one message is sent and received at a time. All PPC services will provide the following modes of I/O:

- Wait-mode I/O - for a read or write operation, control is not returned until the operation completes.
- Non-suspending I/O- for a read or write operation, control is returned immediately to indicate success (for write) or if a message was pending (read).
- Asynchronous I/O - for a read or write operation, control is immediately returned and completion is indicated by receipt of an asynchronous event.

8.7.3.4.2.5 SMC Interface

One of the functions of the SFOC Monitor and Control Subsystem (SMC) is to monitor the status of the SFOC network. The DTS will provide the SMC with enough information to allow the status and load of the network to be analyzed in near-real time. This information will allow analysis of different traffic types (virtual circuit, datagram) and of traffic between specific end points. The DTS will also allow the SMC to enable and disable connections to any node in the SFOC network.

8.7.3.4.3 Common Data Access Subsystem (CDA)

The Common Data Access Subsystem (CDA) provides other SFOC subsystems with a collection of capabilities to aid in the storage, retrieval, and manipulation of SFOC multimission data. The CDA will allow expansion of the SFOC system data requirements without affecting the operation of existing flight projects. The CDA will isolate vendor-specific data management interfaces from applications, thereby allowing the underlying software to change without requiring modifications to applications.

The CDA provides data access capabilities via a set of libraries and interactive utilities. Each is described in more detail in the following sections.

8.7.3.4.3.1 Data Access Libraries

The Data Access Libraries provide a programmatic means of accessing mission data in the following four manners:

- Data Base Management System (DBMS) - allows query access to data in a Relational Data Base Management System. This includes execution of Standard Query Language (SQL) commands.
- The Byte Stream Record I/O - allows access to data in a byte-oriented (traditional UNIX) manner.
- The Spooler I/O file system (JPL unique) - allows extremely fast access to variable length SFDU records. A spooler file is conceptually the same as a circular queue.

- The Record Index Access Method or keyed access method - allows record oriented processing for variable and fixed length records. This includes sequential and indexed access.

The Data Access Libraries allow use of all access methods for both local and remote access to data. In the instance of local use, access is directly through library functions. For the more common instance of remote access, the DTS subsystem is used to establish a virtual circuit with the CDA. The local application will then connect with the appropriate session manager on the remote CDA node. A different session manager will exist for each of the four methods of data access. Once the connection is established, commands and data are exchanged between the local application and the session manager. The session manager accesses requested data by making the appropriate library calls. This combination of local and remote capabilities allows applications to access data in a manner which is indepdenent of its location.

All access to data will be through standard interfaces. This simplifies program maintenance, enhances program portability, reduces software development efforts, and allows for the separation of programming skills.

For each of the four access methods provided by the CDA, the following features are supported:

- Wait, non-suspending, and asynchronous modes - provide different modes of data access. These modes correspond to those described for process to process communication for the DTS subsystem.
- Data location independence - allows applications to be independent of where the accessed data is actually stored.
- Implementation independence - allows applications to be independent of the machine they are running on.
- Security - provides the protection from unauthorized users accessing and/or updating data.
- Concurrent access - allows multiple users to simultaneously access data.
- User exits - allows execution of a user function on the data node before the data is actually saved or returned across the network.
- SFOC Monitor and Control support - allows the SMC subsystem to monitor performance of data access and to regulate access to data.

The CDA also provides catalog services in which the user can determine the location and contents of information on the local node and in the network. Both local and central catalogs are supported. The central catalog is present in the Central Database Subsystem (CDB). This catalog may be queried to obtain the location of all data in the system.

8.7.3.4.3.2 Utilities

Utilities are stand-alone functions which directly provide services to the user. Utilities allow data access without requiring program development. Most utilities are executed from the command line, while a few others may be invoked from applications. The utilities provided include:

- Backup/recovery utilities - used to create a backup of a file to a secondary device. Also allows restoration of previously saved files.

- Byte stream access utilities - provides users with an interactive facility to access files in a byte-oriented manner.
- Record access utilities - provides users with an interactive facility for accessing files in a record oriented manner.
- Spooler utilities - provides users with an interactive facility to access spooler files.
- Data item access utilities - provides users with an interactive facility to perform typical operations on a database file.
- Catalog utilities - provides users with the ability to review and update the local catalog. This utility does not allow update of the central catalog.

8.7.3.4.4 SFOC Monitor and Control Subsystem (SMC)

The SFOC Monitor and Control Subsystem (SMC) is the central point for system monitor and control functions. The SMC will monitor the TIS, CMD, EUA, GIF, CDA, CDB, DTS, and DTV subsystems. The SMC will control the TIS, CMD, EUA, GIF, CDB, DTS, and DTV subsystems. The listed subsystems are involved in the critical downlink and uplink data paths. These data paths consist of subsystems, nodes and data flows which are configured and controlled by the SMC subsystem. This control is necessary to insure optimal performance and to allow corrective actions to be taken when problems arise.

The SMC provides a number of monitor and control functions. These functions are summarized in Figure 8-10.

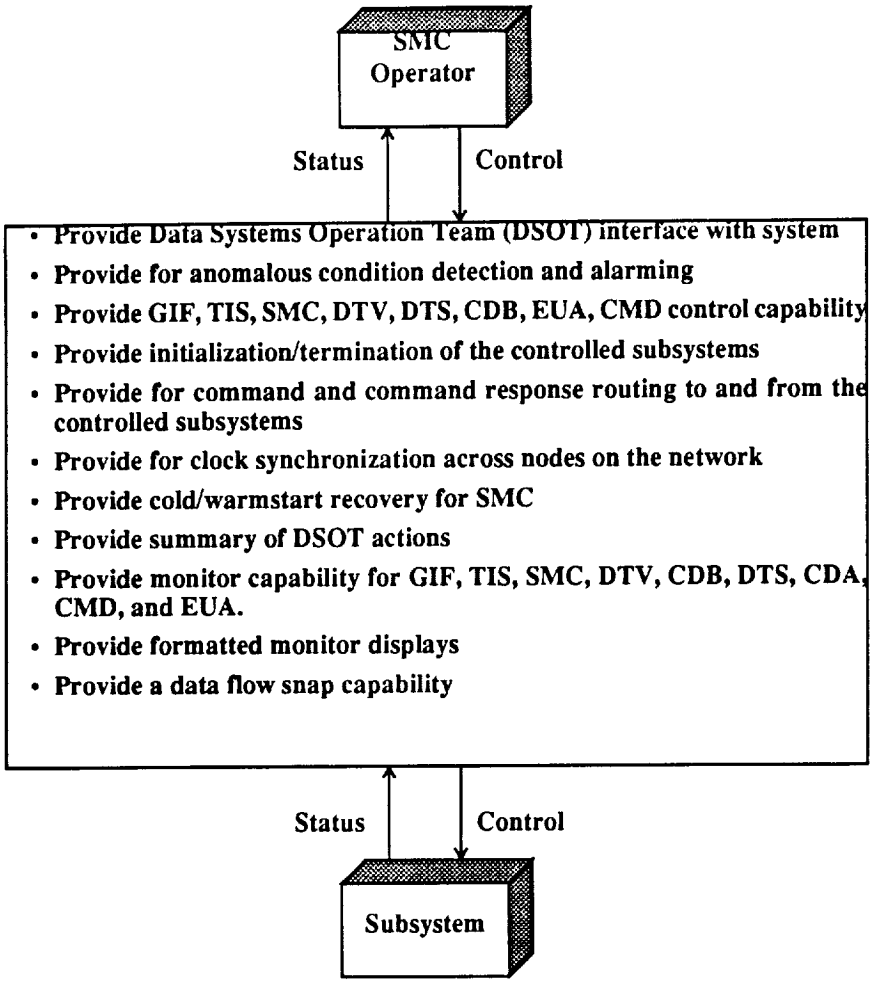


Figure 8-10 SMC Status/Control Functions

The following two sections describe the monitor and control functions of the SMC in more detail.

8.7.3.4.4.1 SMC Monitor Functions

The SMC monitor functions provide the SMC operator with the ability to identify problems with the SFOC data flow. Data flows within SFOC consist of the data itself, processing applications, core subsystems which affect the data flow, the operating system, and the communications network. The SMC monitor functions allow the operator to examine each component of the data flow in an attempt to isolate the problem. The SMC monitor functions are divided into the following three functional areas:

- Subsystem monitor - provides monitor information generated by individual subsystems. This information allows the SMC operator to determine if the data flow problem is due to a particular subsystem. The SMC will monitor status, alerts, events, and snaps (snapshot data samples) generated by individual subsystems.

- System monitor - provides monitor information needed to determine if a data flow problem is the result of a resource used by one of the SFOC subsystems. This includes hardware, networks, and operating systems.
- Monitor display - provides a means whereby the SMC operator can view, manipulate, and interact with the data generated by the subsystem and system monitor functions. Displays are generated by defining templates which specify the required data and its position in the display. Display templates may include filters (time and information type) and logical and relational operators.

8.7.3.4.4.2 SMC Control Functions

The SMC control functions allow the SMC operator to diagnose data flow problems by enabling and disabling any of the subsystems under its control. The SMC operator will also be provided the capability to review and modify the software and hardware configuration which supports the data flow. The SMC control functions are divided into the following four functional areas:

- Application control - provides the ability to issue control directives to any controlled application. The directives provided include start, stop, pause, and resume. This allows restart of failed subsystems and applications. This level of control is available at the local node and remotely from the central SMC node.
- System control - provides a mechanism to maintain system clock times across all nodes on the SFOC network that may be used to process data flows controllable by the SMC operator.
- Control user interface - a single window will be provided for controlling all subsystems, applications, and nodes. The interface will be consistent with that specified by the Workstation Support Environment Subsystem (WSE).
- SMC support tools - provides capabilities required to provide effective analysis of the SFOC system. This includes COTS hardware and software such as LAN analyzers and operating system tools.

8.7.3.5 SFOC Tasks and Major Deliveries

This section discusses the current SFOC tasks which are ongoing or planned for the extended project. This information is provided to allow the reader to gain an understanding of the future directions of the SFOC project. The extended SFOC project plan includes nine distinct development tasks. These include the following:

- System Engineering - involves conversion of all subsystem requirements to the extended project plan and providing planning, engineering, and technical support for all flight projects.
- Baseline Implementation - involves all tasks for the baseline implementation (this includes the adaptation to support the Magellan flight project). This task is responsible for sustaining all SFOC subsystems, adapting subsystems to Magellan, and developing several new subsystems which are to become part of the baseline. These include the Command, Simulation, and External User Access subsystems.
- Flight Projects Office Information Systems Testbed (FIST) - represents a prototyping and evaluation, consulting, and advanced studies group. This group investigates new technology useful for introduction into the SFOC system.

- Adaptation of the SFOC baseline for Mars Observer - involves Mars Observer adaptation and addition of several new capabilities including an updated Telemetry Input Subsystem for processing of multimission data, support and transfer of large data volumes, an external user access subsystem for access to data, stand-alone decommutation for remote users, and support for the sequence database. This flight project will also support Science Operations Planning Computers (SOPCs) which support data access for the remote science community.
- Conversion for Galileo, Voyager, and Ulysses - represents the adaptation of the baseline subsystems to transition these flight projects to the SFOC environment.
- Magellan High Rate - represents the task of supporting the Synthetic Aperture Radar (SAR) device which will be used by Magellan to obtain detailed data for the surface of the planet Venus. The Magellan High Rate (MHR) subsystem will provide this capability.
- Uplink Tool Development - this task provides a set of multimission tools to support uplink communications. This task will support development of sequences which control the activity of a given spacecraft.
- MIPS Transition to SFOC - this task involves the transition of the Multimission Image Processing Subsystem (MIPS) into the SFOC environment.
- NAIF/SPICE Development - the Navigation Ancillary Information Facility (NAIF) SPICE kernel allows complete processing of geometric and other ancillary data obtained from spacecraft. SPICE is an acronym derived from the first letters of the data sets for which it allows processing:
  - Spacecraft ephemeris.
  - Planet, satellite, or other target body ephemeris and associated physical and cartographic constants.
  - Instrument data.
  - C-matrix.
  - Events, as in sequence of events.

  The SPICE kernel will be integrated into the SFOC environment to allow processing of these types of data.

The functionality identified by these 9 tasks will be introduced to the SFOC environment via a series of deliveries. At the current time, two SFOC deliveries have already taken place. Delivery 1 was an internal delivery intended to test the process. Delivery 2 represents the current baseline SFOC system which was used to support the Magellan launch. The current extended SFOC project plan covers the following seven additional deliveries:

- Delivery 3 "Galileo Conversion Demonstration" - provides the first demonstration of the SFOC's multimission capability to support an existing flight project. This delivery will support basic telemetry processing capability for low-rate engineering and science data.
- Delivery 4 "Magellan High-Rate (MHR)" - supports processing for the Magellan Synthetic Aperture Radar (SAR) Instrument (the so-called high-rate telemetry processor). The MHR subsystem will provide support for high-performance processing of the throughput requirements of the Magellan high-rate system.

- Delivery 5 "Mars Observer Flight Sequence Test Support" - supports the flight sequence test for uplink and downlink of Mars Observer. Included are adaptation of the TIS, GIF, simulation (SIM), and DTV subsystems.
- Delivery 6 "Consolidated Conversion Phase 1" - provides the preliminary capability to support low-rate engineering processing for Voyager and Galileo in the SFOC environment.
- Delivery 7 "Completion of Consolidated Conversion" - will comprise the completion of the conversion of Galileo, Voyager, and Ulysses flight projects to the SFOC environment. The command (CMD) and simulation (SIM) multimission subsystems will also be completed for this delivery.
- Delivery 8 "Mars Observer Launch Ground Data System" - will provide the capabilities to support Mars Observer launch and operations. Included are final versions of the Sequence, Command, and External User Access subsystems, and the Science Operations Planning Computers (SOPCs).
- Delivery 9 "Mars Observer Encounter Ground Data System" - will provide the capabilities to support the Mars Observer encounter operations.

8.7.4 Applicability to the Concept Executive

It is difficult to apply the SFOC system to the Concept Executive for two reasons. First, all available documentation for the SFOC system is making the transition to the new project plan. Also, it was not possible to obtain the detailed software specifications for the core subsystems which are of the most interest. Second, the SFOC is an extremely large and complex system whose scope is far beyond that of the Concept Executive.

The SFOC system specifies a baseline which is adapted to support different flight projects. This is the same basic goal of the Concept Executive, in which the concept provides the basic environment for different control centers. The SFOC system and the Concept Executive also share many high-level goals such as expandability, interoperability of hardware and software, and use of COTS components.

The SFOC system is heavily based on use of workstations in a distributed processing environment. The Workstation Support Environment (WSE) and the Data Transport Subsystem (DTS) combine to specify a functional baseline which is similar to that specified by the Concept Executive.

The primary command interface provided by the WSE appears to be the UNIX shell (as opposed to an environment specific language). This is indicated by the various WSE functional requirements which explicitly require UNIX C shell-like functionality.

The SFOC system is dependent on the use of standards, including UNIX, X Windows, C and FORTRAN, and TCP/IP. This is again similar to the Concept Executive, but with less detail devoted to specification of individual standards. For example, which higher level user interface standards have been selected (if any).

The Command Data Access Subsystem (CDA) provides an interesting concept which could be applied to the host systems which provide non-real-time data to Concept Executive applications. A standard interface which allows multiple access methods to near real-time and archived data would be useful in the Concept Executive. The Concept Executive specifies a standard data acquisition process, but it offers less functionality than defined by the SFOC CDA subsystem.

The SFOC Monitor and Control Subsystem is interesting as it provides a centralized manner in which all workstations, the SFOC network, and applications are monitored and controlled. While this function may not be useful within the Concept Executive, a system of this type is definitely required within the different control center environments. An integrated approach to all monitor and control functions is interesting and may prove more efficient than separate systems for network and workstation status and control.

The SFOC system does not provide an automated configuration management plan. A configuration management plan is described in one of the "plan" level documents, but is primarily expressed in terms of procedures and responsibilities.

9.0 User Interfaces

This chapter summarizes two user interface systems available from different organization. The two systems were reviewed due to their potential usefulness for the Concept Executive. The systems reviewed (and the sources) include the following:

- Transportable Applications Environment (TAE) - Goddard Space Flight Center.
- Motif - Open Software Foundation (OSF).

Review of each system was obtained by experimenting with the actual software and reviewing the associated documentation. An on-site demonstration of TAE was also provided. The following sections describe each system in more detail.

9.1 TAE System

The Transportable Applications Environment (TAE) is a portable User Interface Management System (UIMS) which provides an integrated environment for designing, building, prototyping, and tailoring an application's User Interface (UI). The TAE also provides effective management of the UI throughout the application's execution. It was developed by Century Computing, Inc. for NASA Goddard Space Flight Center. The TAE is a collection of callable subroutines, development tools, and run-time libraries that provide and support a consistent user interface for interactive systems.

At run-time, TAE applications sit between the user and the system application tasks, acting as the user interface executive by displaying menus, command line prompts, help information, and messages, and by passing parameters from the user to the applications.

Based on continually refined user requirements, TAE has become a powerful tool for quickly and easily building consistent, portable user interfaces in an interactive environment. TAE can be used to generate much of an application system's user interface, resulting in reduced system development time. Since a major goal of TAE is consistency, the user interface of an application system remains consistent from task to task. Also, once a user is familiar with the TAE user interface, using any system developed under TAE becomes much easier. Finally, since TAE user interfaces are easy to define and can be completely separate from the application software, rapid prototyping of the user interface is possible.

TAE comes in two forms. The original TAE, called TAE Classic, uses a standard ASCII terminal. TAE Plus, which includes TAE Classic, supports modern workstations with a window-based, mouse and keyboard driven iconic interface. Our review was done on the TAE Plus version. TAE Plus evolved because of the rapid emergence of sophisticated workstations with high-resolution, bit-mapped displays. These systems employ multiple windows, graphical icons and objects, color, and various input devices and techniques to develop intuitive, friendly interfaces that efficiently display complex information. TAE Plus provides the application developer a method of working with the modern user interface capabilities made available by the new family of graphic workstations.

9.1.1 Contact Point

The TAE system is distributed by Century Computing, Incorporated for NASA Goddard Space Flight Center. It is publicly available to interested organizations. To obtain a copy of the software and documentation, contact:

NASA Goddard Space Flight Center

TAE Support Office

Greenbelt, Maryland 20771

9.1.2 Review Process

The review process for the TAE system involved installation and use of the 4.0 version of the software on a Sun 3/60 system. In addition, the following documents were reviewed:

- Introduction to TAE.
- Overview of TAE Plus.
- TAE Plus User Interface Developer's Guide.

9.1.3 System Description

The TAE system provides a suite of integrated tools and software libraries for developing and running highly interactive, graphical application systems. TAE Plus includes:

- TAE Plus WorkBench.
- Window Programming Tools (WPT).
- TAE Plus applications executive, TM.
- WPT TAE Command Language (TCL) prototyping commands.
- Facelift, a graphical interface to TM.

TAE Plus is based on the X Window System, Version 11, Release 3 (X11), and on the X Toolkit, both emerging standards in graphical window-based software. The WPT routines provide a buffer between an application and the complexities of X11 and the X Toolkit, but an application may also access the X Toolkit and Xlib directly.

The application code in TAE is generated by the WorkBench and enhanced by the developer. The generated application will include calls to the WPT library that define and control TAE Plus panels and interaction objects. The X Toolkit provides the building blocks used by WPT to define panels and interaction objects. Xlib defines the X11 graphics primitives and the X11 server provides the interface and the display hardware.

The TAE Plus WorkBench is a development tool that supports the interactive design and layout of graphical, panel, and interaction object-based user interfaces. TAE Plus panels are similar to windows. The TAE Plus interaction objects are display objects, such as buttons, scrolling lists, pull-down menus, text input fields, and dials, that display information for, and receive information from, the user.

The Window Programming Tools (WPT), is a package of application-callable routines used to display and control an application's user interface. Using WPT, an application can create, display, modify, and delete the graphical attributes of, and receive input from, TAE Plus panels and interaction objects.

TAE Plus includes an applications executive called the Terminal Monitor (TM). TM supports an interpreted command language called the TAE Command Language (TCL), and provides various executive-type user functions for suites of applications, including session logging, application sequencing, and both command line and menu-based user modes.

A set of TCL commands, called WPT TCL commands, mirror the graphical WPT routines described above. These commands allow a developer to create graphical TAE Plus applications us-

ing only TCL. Because TCL is interpreted, and thus can be executed without compiling and linking, this capability supports rapid initial prototyping of user interfaces.

The standard menus, help displays, and parameter entry displays of the applications executive, TM, are typically ASCII-based. However, by enabling the *Facelift* feature of TM from a graphics terminal, these standard displays are "facelifted" with a graphical look and feel that is consistent with TAE Plus WPT-based applications. This enables development of a user interface that will operate on ASCII-based terminals and/or graphics workstations.

9.1.3.1 User Environment

The TAE Plus user interface has four categories of interface elements:

- Panels (windows).
- Interaction objects.
- Command line.
- Formatted screens: menu, parameter, and help.

TAE Plus offers a set of panels (windows) and interaction objects for constructing graphical, interactive applications. A panel is a window of information to be displayed. It has an optional title bar. A workspace panel is a special panel for the display of application graphics not supported by the WPT routines. Data driven objects are available for representing dynamically changing data. The object type describes the behavior that will be exhibited, e.g., rotation, moving, stretching. The images used for these objects may be defined by the designer using a graphic editor accessible through the WorkBench. After drawing the images, the designer names the dynamic and static background portions and describes information needed for the behavior of the object, such as range of movement.

TAE Plus provides a command line interface through the TAE Command Language (TCL). The command line mode allows an end user to interactively enter TCL commands that are interpreted by the Applications Executive (TM).

A set of standard screens -- menu, tutor, and help -- are available for alphanumeric terminals and graphics workstations when running with TM. A menu is a display containing a set of choices, each of which corresponds to another menu or a TCL command to be executed. Tutor mode assists the user in specifying the parameters for an application. For each parameter, the tutor display presents the parameter name, some descriptive text, and the current value, if one exists. A help screen is always available and there is a standard display format for the help text.

9.1.3.2 Development Environment

TAE Plus offers an interactive graphical environment for creating applications, where the interface construction is separated from the application logic. This separation makes it possible to do iterative changes to the interface without changing the application code. The TAE Plus developer's environment consists of a powerful interactive graphical WorkBench and facilities for rapid prototyping and generating skeletal code for an application in several languages.

At development time, the developer interacts with TAE Plus WorkBench to create a user interface. Using the WorkBench, the developer creates a resource file and application code. The resource file includes the definitions of the panel and item attributes that make up the interface. The application code includes logic to open the resource file at run-time, extract the panel and item attributes, and display them as required.

The TAE Plus WorkBench is the primary tool of the development environment. Through the WorkBench an interface designer can compose an interface from interaction objects, the basic building blocks for an interface. The designer can create, size, move, and connect objects and change their attributes. When the interface design is saved, a resource file is created that describes the interaction objects and connections in the interface.

Developing a user interface is typically an iterative process. Once a user interface has been initially created and stored in a resource file, the WorkBench can read in that resource file and the user interface can be modified and saved again in an updated resource file. These modifications can often be made without changes to the application program.

9.1.3.2.1 WorkBench

The main component of TAE Plus is the "WorkBench" which is an application development tool that supports the interactive definition and layout of sophisticated user interfaces. The WorkBench allows an application developer to interactively construct the look and feel of an application screen by arranging and manipulating interaction objects. The interface to the WorkBench allows both programmers and non-programmers to easily design complex user interfaces.

The WorkBench allows a developer to build an interface as if it were composed from a construction set of components that have visual attributes such as color, font, size, and location which can be tailored. The basic building blocks for developing an application's user interface are a set of interaction objects. All visually distinct elements of a display that are created and managed using TAE Plus are considered to be interaction objects. There are three categories of interaction objects within TAE: user-entry objects, information objects, and data-driven objects. User-entry objects which are mechanisms by which an application can acquire information and directives from the end user. Information objects which are used by an application to instruct or notify the user. Data-driven objects which are vector-drawn graphic objects which have been "connected" to an application data variable, and elements of their view change as the data values change.

Functionally, the WorkBench allows a user interface developer to dynamically lay out an application screen, defining its static and dynamic areas. From within the WorkBench, a developer can:

- Define, position, re-size, move, copy, and set the display attributes and default values of interaction objects and panels.
- Define user interface connections that connect the user selection of an interaction object, such as a button, with the deletion of that button's panel and/or the display of another panel.
- Use a bitmap editor to create graphical icons.
- Draw the images for data driven objects.
- Immediately apply the changes to the look of any item.
- Undo the last action.
- Specify when panels will appear and disappear in the application.
- Rehearse the user interface.
- Generate skeletal application code, in either C, Ada, or the TAE Command Language (TCL), which displays and controls the designed user interface.

- Save the user interface separately from the application code (thus enforcing separation of "form from function") in a TAE Plus resource file.
- Modify an existing user interface, often without any change to the application code.

The general style of interaction when using the WorkBench is one of point and select. WorkBench panels and the newly created application panels coexist on the display. The WorkBench panels have a consistent and unobtrusive look so that they are easily distinguishable from the designer's application panels. Help is associated with every panel in the WorkBench and is accessed by selecting the "?" icon in the upper right corner of the panel.

Once an item is created, it appears on the display and can then be moved or re-sized. The developer specifies the user interface through menu selection and minimal typing. Whenever possible, functions apply to the currently selected interface object. For instance, a selected object is duplicated when the copy command is selected; a panel name in a WorkBench panel will be filled in when the panel is selected.

Most of the panels the designer sees are of three types: the main panel, a specification panel, and a presentation panel. The main panel provides the top level functions for the WorkBench. It allows the basic file, editing and utility functions through pull-down menus. The file commands include saving and retrieving files. The edit commands include modifying, aligning, duplicating, and specifying the initial panel. The utility functions include starting the window manager, opening a terminal window, generating code, rehearsing the interface, and browsing through a list of the available fonts.

From the main panel, the WorkBench may be operated in three modes. The first mode, move/resize/edit, is the default for the WorkBench. It allows the designer to create and design the application panels and items. The second mode is the Set Default Values. This mode allows the designer to set default values of interaction objects. The third available mode is the Connections mode which allows the designer to establish an action for each user event.

The specification panels are used to define and modify panel and interaction objects, called items. The WorkBench contains a panel for specifying panels and items in the user interface. These panels request information which is required to define the object. TAE Plus also offers an optional help feature which provides a consistent mechanism for supplying application-specific information about a panel and any interaction items within the panel.

The presentation panels are displayed when the Details button is selected in the Item Specification Panel. This panel solicits details on the behavior specific to the interaction object selected.

Once a user interface has been developed and tailored, the WorkBench provides a "generate" function which produces a fully operational and commented body of code which will display and manage the entire UI. Currently, source code generation of C, Ada, and TCL is supported, with bindings for Fortran and C++ expected in later TAE Plus releases.

9.1.3.2.2 Window Programming Tools

Once the application's screen has been designed, the WorkBench saves the user interface details in a resource file. TAE Plus includes runtime services, Window Programming Tools (WPTs), which are used by application programs to display and control the user interfaces designed with the WorkBench. The WPT package is a library of application program callable routines that are used to define and control elements of the TAE Plus user interface. Using WPT routines, applications can:

- Display and erase panels and their associated interaction objects.
- Update the displayed value of an interaction object.
- Reset the value of an interaction object to its initial value.
- Update the presentation attributes of an interaction object.
- Get the next panel-related event.
- Temporarily display a busy cursor.
- Display a message in a "bother box" (i.e., error messages).

WPT applications must run on a graphic workstation that supports the X Window System, Version 11, Release 3. The WPT routines will not execute on ASCII alphanumeric terminals.

9.1.3.2.3 TAE Command Language

In addition to providing the WPT runtime subroutines, TAE Plus also offers control of interaction objects throughout the interpreted TAE Command Language (TCL). This capability provides an extremely powerful means to quickly prototype an application's use of TAE Plus interaction objects and add programming logic without the requirement to compile or link. The TAE Command Language (TCL) is a procedural prototyping language which is interpreted by the TAE Plus applications executive, TM.

TCL offers a high-level set of commands used to invoke and manage application functions. Commands can be invoked dynamically during an interactive session or used to build command procedures. The basic unit of execution in TCL is the command string. A TCL command string may be a TCL built-in command or a procedure invocation. Procedures may contain invocations of other procedures. The TCL built-in commands include a set of WPT TCL commands that are analogous to WPT calls and allow for access to the graphic objects in TAE Plus.

The TCL commands are analogous to the WPT routines. As with WPT routines used by application programs, WPT TCL commands can be used to directly define panels and interaction objects, or they can be used to access WorkBench-generated object definition files that contain pre-defined panels and interaction objects.

TCL has the following general capabilities:

- Local and global variables.
- Variable assignment and expressions.
- Macro-level substitution for variables and parameters.
- Constructs for conditional execution and looping.

TAE Plus obtains a command string from a command line source, i.e. via a terminal window, and determines whether the string is a TCL intrinsic command or a procedure invocation. If the string is a procedure invocation, TAE Plus locates and executes the procedure.

9.1.3.3 Run-time Environment

There are two modes for running an application under TAE Plus: with and without the application executive, TM. Applications designed via the WorkBench using WPT calls may be run without TM. A C or Ada application program accesses the resource file at run-time. The WPT routines affect the display of the user interface stored in the resource file. TM manages applications running on both graphics and ASCII terminals. By running under TM, these applications can receive pa-

rameters collected by TM, and can send parameter values back to TM. TM interprets TCL commands from either a command line in a terminal emulator window or from an ASCII terminal, or from a TCL procedure.

The TCL user interface code automatically generated in the WorkBench displays and interacts with panels and items via a set of commands that are analogous to the Window Programming Tools (WPT) routines. These WPT TCL commands are a subset of the TCL commands that are used to directly define panels and interaction objects.

While the WPT application accesses the workstation display through panels and interaction objects, TM may still access an alphanumeric terminal to display menus, help, parameter entry screens, and the command line. Using *Facelift,* TM menus, help screen, and parameter entry screens can be displayed using a panel and interaction object interface on the workstation display.

9.1.4 Applicability to the Concept Executive

TAE Plus supports user interface prototyping through its rehearse mode and generate function in the WorkBench. The rehearsal mode allows the user to execute the designed user interface as if it were operational for the end user. The generate function will then take the designed prototype of the user interface and generate a skeleton program which will display and control the designed user interface. This generated code can then be executed as is or it can be integrated with application code to create a complete application system.

TAE is only applicable to the Concept Executive in a limited way. Since this system is only concerned with the design and development of user interfaces that aspect of an executive is the only area where the TAE could be used for its function. The TAE package has many positive features which could be implemented in the executive. The WorkBench used by TAE is an exceptional tool for prototyping a user interface. It allows a system developer to quickly create a user interface and rehearse all of the events and actions which will cause the various defined panels to be displayed. The nice thing about the WorkBench is that once the interface has been designed and rehearsed, it can be used to generate the application code to execute the interface. Currently, TAE will generate C, Ada, or TCL code and is designed to eventually generate C++ code for those who work with that environment.

Another important feature of TAE is that it is based on the X Windows standard. In this age of technology and advancement, the use of standards in the programming world is becoming an undeniable requirement for large systems. The use of X Windows helps achieve transportable code. New releases of TAE will use the X Toolkit and the Hewlett Packard public-domain widget set. This provides an excellent user interface development environment. TAE is also planning to migrate to the Motif standard in the near future. The extent of this migration is not known at this time. Motif includes a user interface language which allows user interfaces to be specified without programming. To fully utilize Motif, TAE would have to generate user interface language statements (as well as C and other languages).

One of the major functions of the TAE system is the Data Driven Objects (DDO's). In theory, DDO's could be used to display telemetry data. However, the more animated DDO's would probably involve too great a processing burden to be used for large amounts of data. It would be worthwhile for the Concept Executive to provide a DDO-like widget for the display of real-time data values.

TAE is a viable tool for creating user interfaces because it allows the user (developer) to use the WorkBench to generate their interface code and then allows them to add to the code, integrating the user interface code with other application software. It also provides the user with an intermediate set of X interface calls (WPT) or will allow the user to access the X library directly.

By using TAE the developer will be able to create a user interface which is consistent across the system. If developers are left to their own devices, the user interfaces across the different disciplines of the system will have more likelihood of being inconsistent. Of course, even with TAE, the placement of items in a panel is completely up to the developer. For this reason, there should still be a set of user interface guidelines or standards established which must be used for every user interface developed in the system (i.e., the placement of a help button in the upper right corner of every panel).

9.2 OSF/Motif

OSF/Motif was developed jointly as a Unix user interface standard by Hewlett-Packard, Microsoft, and Digital Equipment Corporation for the Open Software Foundation (OSF). It is a base for creating a consistent and easy-to-use graphical user interface on systems from multiple vendors. To support various hardware platforms, OSF/Motif uses the MIT X Window System protocol standard. Motif is delivered as a combination of components - a User Interface (UI) toolkit, a window manager, and a UI description language. Using Motif, application programmers can build the basic objects of their UI, such as buttons, pull-down and pop-up menus, and scroll bars in a way which will provide the user with a consistent look and feel for the interface.

9.2.1 Contact Point

The Motif system is the first product distributed by the Open Software Foundation, a not-for-profit Unix software distributor. To obtain a copy of the software and documentation, contact:

Open Software Foundation

Eleven Cambridge Center

Cambridge, MA 02142

9.2.2 Review Process

Motif was reviewed by porting a set of UI applications written using X Windows and the Athena toolkit to the Motif toolkit. During this process the following documentation was referenced:

- Programmer's Guide / Toolkit.
- Programmer's Guide / Motif Window Manager.
- Programmer's Guide / User Interface Language.
- Programmer's Reference Guide.

9.2.3 System Description

OSF/Motif is a graphical user interface based on the MIT X Window protocol. The OSF/Motif environment is based on an object-action input selection model. The selection model defines the actions that users must perform to control the window manager and applications in the OSF/Motif environment. The selection model follows a point-and-click paradigm. Users first point at and select an object with which to work, and then point at and select an action to perform on the selected object. OSF/Motif provides the following:

- Toolkit.
- Window Manager (MWM).
- User Interface Language (UIL).
- Style Guide.

The OSF/Motif toolkit is a rich and varied collection of widgets and gadgets for building OSF/Motif applications. The toolkit provides a standard graphical interface upon which the window manager is based. Toolkit widgets provide a 3-D reference appearance that gives users visual cues to the effects of their actions.

The User Interface Language (UIL), the OSF/Motif presentation description language allows application developers and interface designers to create simple text files which describe the visual properties and initial states of interface components. Changes to components are made in the text file, eliminating the need to change application code when tuning an interface.

The window manager works with the toolkit to manage the operation of windows on the screen. The window manager provides functions for moving and resizing windows, reducing windows to icons, restoring windows from icons, and arranging windows on the workspace. An additional OSF/Motif window manager feature is the icon box. The icon box contains icons for all windows operating under the window manager. The window manager provides users with a way to manipulate the windows displayed in their OSF/Motif environment.

The OSF/Motif package also comes with a style guide which describes the standards for window manager and toolkit behavior. This guide provides application writers with guidelines for using toolkit widgets, widget writers with guidelines for designing new widgets, and window manager writers with guidelines for designing new or customized window managers.

9.2.3.1 Toolkit

The OSF/Motif widget set is based on the Xt Intrinsics, a set of functions and procedures that provide quick and easy access to the lower levels of the X Window system. The Motif Widget system is layered on top of the Xt Intrinsics, which in turn are layered on top of the X Window System, thus extending the basic abstractions provided by X. The Motif Widget system supports independent development of new or extended widgets. The Motif Widget system consists of a number of different widgets, each of which can be used independently or in combination to aid in creating complex applications. Applications can be written faster and with less lines of code using the Motif widgets; however, they will require more memory than similar applications written without using these widgets.

In Motif, every widget is dynamically allocated and contains state information. Every widget belongs to one of many classes, and each class has a structure that is statically allocated and initialized and contains operations for that class. The hierarchy of the basic widget classes in Motif are presented in Figure 9-1.

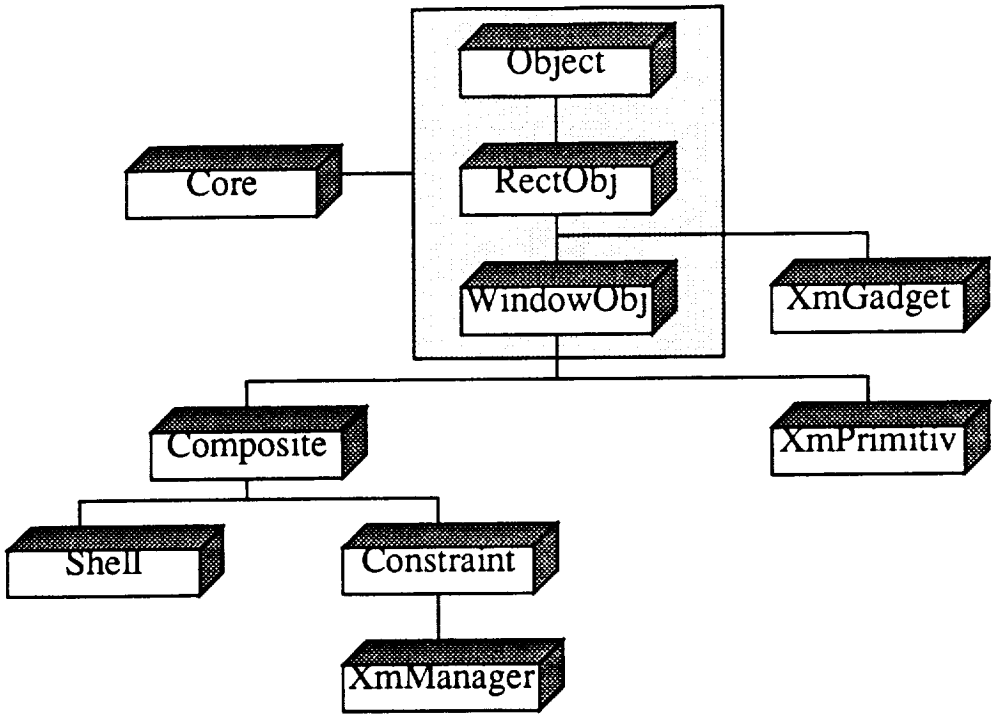


Figure 9-1 Motif Widget Hierarchy

The basic class is the Core class. It contains resources that are inherited by all other classes. Two classes are layered beneath the Core class, the Composite class and the Primitive class. The Primitive class has no other classes beneath it, but the Composite class has two - the Constraint class and the Shell class. Each lower class can inherit some or all of the resources belonging to a higher class. For example, a Manager class widget can inherit some or all of the resources belonging to the Constraint class, the Composite class, and the Core class.

Motif has a variety of widgets and gadgets, each designed to accomplish a specific set of tasks, either individually or in combination with others. There are also convenience functions that create certain widgets or sets of widgets for a specific purpose.

Widgets are used either individually or in combination to make the creation of complex applications easier and faster. An instance of a widget class is composed of a data structure containing values and procedures for that particular widget instance. There is also a class structure that contains values and procedures applicable to all widgets of that class.

Widgets are grouped into classes according to the function of the widget. Logically, a widget class consists of the procedures and data associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses. Physically, a widget class is a pointer to a structure. The contents of this structure are constant for all widgets of the widget class. A widget instance is allocated and initialized in OSF/Motif by **XmCreate<widget name>, XmCreateWidget,** or **XmCreateManagedWidget**. The OSF/Motif documentation divides the widgets into five categories:

- Shell Widgets.
- Display Widgets.

- Container Widgets.
- Dialog Widgets.
- Menu Widgets.

9.2.3.1.1 Shell Widgets

Shell widgets are top-level widgets that provide the necessary interface with the window manager. Different Shell widget classes are provided for the various categories of top-level widgets. The Xt Intrinsics provide some underlying shells and the Motif toolkit provides the remaining shells.

The Xt Intrinsics provide the following shell classes:

- Shell - base class for shell widgets. It provides resources for all other types of shells. This class is internal and cannot be instantiated.
- Override Shell - used for shell windows that completely bypass the window manager.
- WMShell - contains resources that are necessary for the common window manager protocol. This class is internal and cannot be instantiated.
- Vendor Shell - contains resources used by vendor-specific window managers. This class is internal and cannot be instantiated.
- TransientShell - used for shell windows that can be manipulated by the window manager but cannot be iconified.
- TopLevelShell - used for normal top-level windows.
- ApplicationShell - used for an application's top-level window.

The Motif toolkit provides the following widgets:

- XmDialogShell - used as the parents of modal and modeless Dialogs associated with other top-level windows.
- XmMenuShell - used as the parents of MenuPanes.
- VendorShell - provides the common state information and services needed by the window-manager visible shells.

9.2.3.1.2 Display Widgets

Display widgets are widgets that provide a means for displaying text in an non-editable fashion. The Display widgets include:

- Core class - provides common resources that are needed by all widgets, including x and y location, height, width, window border width.
- XmPrimitive -provides resources for things such as border drawing and highlighting, traversal activation and deactivation.
- ArrowButton - consists of a directional arrow surrounded by a border shadow. The ArrowButton will change its shadow appearance to indicate whether the button has been selected (pressed in) or deselected (released).
- DrawnButton - consists of an empty widget window surrounded by a shadow border. It provides the application developer with a graphics area that can have PushButton input semantics.

- Label - consists of either text or graphics. Label's text is a compound string and can be multi-directional, multi-line, multi-font, or any combination of these.
- List - allows the user to make a selection from a list of items.
- PushButton - consists of a text label or pixmap surrounded by a border shadow. PushButtons are used to invoke actions, such as run, cancel, stop, and so on.
- ScrollBar - allows you to view data that is too large to be viewed in its entirety. ScrollBars are combined with a widget that contains the data to be viewed.
- XmSeparator - a primitive widget to be used as an item separator placed between items in a display.
- XmText - provides a single or multi-line text editor that has a user and programmer interface that can be customized. It can be used for single-line string entry, forms entry with verification procedures, multi-page document viewing, and full-screen editing.
- XmToggleButton - consists of a text or graphics button face with an indicator (a square or diamond shaped box) placed to the left of the text or graphics.

9.2.3.1.3 Container Widgets

Container widgets are Composite widgets that provide applications with general layout functionality. Since they are Composite widgets, Container widgets can have children. All of the container widgets are built from the Core, Composite, Constraint, and XmManager widget classes. Motif provides the following container widgets:

- XmManager - acts as a supporting superclass for other widget classes. It supports the visual resources, graphics contexts and traversal resources necessary for the graphics and traversal mechanisms.
- DrawingArea - an empty widget that is easily adaptable to a variety of purposes. DrawingArea does no drawing and defines no behavior except for invoking callbacks to notify the application when graphics need to be drawn and when the widget receives input from the keyboard or mouse. It supports minimal geometry management for multiple widget or gadget children.
- XmFrame - a manager that is used to enclose a single child within a border drawn by the XmFrame widget.
- XmMainWindow - provides a standard layout for the primary window of an application. This layout includes a MenuBar, a CommandWindow, a work region, and ScrollBars. Any or all of these areas are optional.
- RowColumn - a general purpose RowColumn manager capable of containing any widget type as a child. It requires no special knowledge about how its children function and provides nothing above and beyond support for several different layout styles. The type of layout can be configured to lay out its children in either a row or a column fashion and can specify whether the children should be packed tightly together, symmetrically, or with specific x and y positions.
- Scale - is used by an application to indicate a value from within a range of values and allows the user to input or modify a value from the same range. A Scale widget allows you to select a value from a range of displayed values by adjusting an arrow to a posi-

tion along a line. A Scale has an elongated rectangular region with a slider that is used to indicate the current value along the Scale.

- ScrolledWindow - combines one or more ScrollBar widgets and a viewing area to implement a visible window onto some other (usually larger) data display. It can operate in an automatic manner where it performs all scrolling and display actions or it can provide a minimal support framework.
- PanedWindow - is a Composite widget that lays out children in a vertically tiled format. Children appear from top-to-bottom, with the first child inserted appearing at the bottom.

9.2.3.1.4 Dialog Widgets

Dialog widgets are container widgets that provide applications with layout functionality typically used for popup "dialogs." These widgets are used for interaction tasks such as displaying messages, setting properties, and providing selection from a list of items. Dialog widgets are thus used primarily as an interface between the user and the application.

A Dialog widget will normally ask a question or present the user with some information that requires a response. In some cases the application will be suspended until the user provides the response.

A Dialog is a collection of widgets, including a DialogShell, a BulletinBoard (or subclass of BulletinBoard or some other container widget), plus various children of the BulletinBoard. All of the dialog widgets are built form the Core, Composite, Constraint, and Manager widget classes.

A Dialog can be built by building up the necessary argument lists and creating each individual widget in the Dialog. For common interaction tasks, convenience functions are defined that create the collection of widgets that comprise a particular Dialog. The collections of widgets created by Dialog convenience functions are referred to as Convenience Dialogs.

9.2.3.1.4.1 Dialog Widget Descriptions

Those widgets which are considered the basic dialog widgets are:

- BulletinBoard - provides simple geometry management for children widgets. BulletinBoard is the base widget for most dialog widgets, but is also used as a general container widget.
- Command - includes a command history region and a command input region. Command also provides a command history mechanism.
- FileSelectionBox - used to get a selection from a list of alternatives. FileSelectionBox includes an editable text field for the directory mask, a scrolling list of filenames, and an editable text field for the selected file.
- Form - that provides a layout language used to establish and maintain spatial relationships between its children. Form includes the base level of dialog support and can also be used as a general container widget.
- MessageBox - used to give information to the user. MessageBox includes a symbol and a message.
- SelectionBox - used to get a selection from a list of alternatives. SelectionBox includes a message, and editable text field, and a scrolling list of choices.

9.2.3.1.4.2 Convenience Dialogs

Convenience Dialogs are collections of widgets that can be created by using convenience functions. Each convenience dialog instantiates a dialog widget as a child of a DialogShell. Most of the Convenience Dialogs are created with some default buttons which give the user the options of "OK", "Cancel", and "Help."

Convenience Dialogs are either modal or modeless. A modal dialog stops the work session and solicits input from the user. A modeless dialog solicits input from the user, but does not interrupt interaction with any application.

The Convenience Dialogs provided by Motif are:

- BulletinBoardDialog - used for interactions not supported by the standard dialog set. Necessary dialog components are added as children of the BulletinBoard.
- ErrorDialog - used to warn the user of an invalid or potentially dangerous condition.
- FileSelectionDialog - used to select a file.
- FormDialog - used for interactions not supported by the standard dialog set. Necessary dialog components are added as children of the Form.
- InformationDialog - used to give information to the user, such as the status of an action.
- MessageDialog - used to give information to the user.
- PromptDialog - used to prompt the user for text input.
- QuestionDialog - used to get the answer to a question from the user.
- SelectionDialog - used to get a selection from a list of alternatives.
- WarningDialog - used to warn the user of the consequences of an action, and give the user the choice of resolutions.
- WorkingDialog - used to inform the user that there is a time consuming operation in progress and gives the user the ability to cancel the operation.

9.2.3.1.5 Menu Widgets

The RowColumn widget is the basis for most of the menu system components. It has a built-in ability to behave like a RowColumn manager, a RadioBox, a MenuBar, a Pulldown MenuPane, a Popup MenuPane, and an Option menu. Convenience functions have been provided to easily create these special versions of the RowColumn widget.

The Motif menu system is composed of the following widgets and convenience functions:

- XmRowColumn (Widget).
- MenuBar (Convenience Function).
- OptionMenu (Convenience Function).
- Pulldown MenuPane (Convenience Function).
- Popup MenuPane (Convenience Function).
- XmMenuShell (Widget).
- XmCascadeButton (Widget).
- XmSeparator (Widget and Gadget).

- XmLabel (Widget and Gadget).
- XmToggleButton (Widget and Gadget).
- XmPushButton (Widget and Gadget).

Applications are not required to use all of these components to use the menu system.

9.2.3.1.6 Gadgets

Gadgets provide essentially the same functionality as the equivalent primitive widgets. The primary motivation behind providing a set of gadgets is to improve performance, both in execution time and data space. This applies to both the application and server processes and minimizes the amount of lost functionality. The performance difference between widgets and gadgets is dramatic, so it is highly recommended that applications use gadgets whenever possible.

Gadgets can be thought of as a windowless widget. This means that they do not have windows, translations, actions, or popup children. Also, gadgets do not have any of the visual resources found in the XmPrimitive class for primitive widgets. These visuals are referenced by a gadget from its parent.

Examples of display gadgets include buttons, labels and separators. All of these gadgets are built from the classes of Object, RectObj, and XmGadget. The Gadgets provided by Motif include:

- Object - is an Xt Intrinsics meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other classes.
- RectObj - is an Xt Intrinsics meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other classes.
- XmGadget - is a Motif meta class and is therefore never instantiated. It is used as a supporting superclass to provide common resources to other gadget classes.
- XmArrowButtonGadget - has the same functionality as PushButtonGadget but displays a directional arrow within itself.
- XmLabelGadget - consists of either text or graphics. It can be instantiated but it is also used as a superclass for button widgets.
- XmPushButtonGadget - used to issue commands within an application.
- XmSeparatorGadget - used to provide a visual separation between groups of widgets. It can draw horizontal and vertical lines in several different styles.
- XmToggleButtonGadget - consists of a text or graphics button face with an indicator placed to the left of the text or graphics. ToggleButtonGadgets are used for setting non-transitory data within an application.

9.2.3.1.7 Convenience Functions

Convenience functions are functions that enable you to create certain widgets or gadgets, or groups of widgets or gadgets, by making just one function call. A Convenience function creates a predetermined set of widgets and returns the parent widget's ID. For widgets and gadgets other than Dialog widgets, Convenience functions are of the form:

XmCreate<widget name>.

For Dialogs, convenience functions are referred to as Convenience Dialogs, and are of the form:

<DialogWidgetName>Dialog.

It is very easy to use a convenience function to create a widget. The **XmCreate<widget name>** functions create unmanaged widgets. Your application must manage the set of widgets before they will be displayed. You can manage each widget separately or as a group.

9.2.3.2 Window Manager

The OSF/Motif user interface provides a rich environment, designed to facilitate communications between users and an application. This environment is composed of discrete graphical elements.

The graphical elements of the OSF/Motif user interface facilitate communication by providing users with a "workspace" which provides interaction with an application which is more familiar (thus more intuitive) and less technical than the traditional user interface provided by the command-line prompt. "Workspace" is used in the Motif environment to emphasize that the functionality and graphical elements of the user interface are tools that empower users to accomplish tasks with their computers.

The OSF/Motif window manager (MWM) frames application windows with an eight-segment border that can be stretched to resize the window. A title area supplied by the window manager displays a title for the window and can be used to move it. Graphical buttons embedded in the window manager frame provide a window management menu and other window controls. Additionally, the OSF/Motif window manager has a three-dimensional appearance so that the control buttons, when "pressed" by the mouse pointer, actually look like they have been pressed. The window manager helps provide for consistent behavior from one application to the next.

The elements of the user interface, the objects that users see (for example, windows, icons, menus, and dialog boxes) appear on the workspace and can be stacked one on top of one another like papers on a desk or tools on a workbench.

The OSF/Motif Window Manager (MWM) provides window management facilities within the framework of the OSF/Motif environment. MWM provides you with an industry standard user interface, a high degree of flexibility, and a pleasing visual interface.

MWM facilitates user-computer communications in the following areas:

- MWM provides for direct manipulation of graphic objects using an object-action model. A user controls the operation of an application program by selecting a window, menu, icon, or other graphic object and then indicating an action to be done to that object.
- MWM uses two ASCII configuration files, .Xdefaults and .mwmrc. By editing these files, users can choose the size, locations, and color of the graphic elements in their environments.
- MWM allows keyboard-only access to window management functionality in cases where mouse access is not available or keyboard access is preferred.
- MWM provides a consistent appearance and behavior using the OSF/Motif X Widgets visual style as specified in the OSF/Motif Style Guide.

MWM provides all required elements of OSF/Motif behavior, but it is also extensible so you can modify window appearance and behavior to suit the specific needs of your application.

In conformity with OSF/Motif behavior, MWM will allow a user to perform window management functions without using a mouse. Window management functions performed from the keyboard generally apply to the active window, the one window that is getting keyboard input. However,

you can also use the keyboard for non-specific window management functions such as changing the stacking order of windows on the screen.

While the default window management behavior is recommended for the sake of consistency, OSF/Motif Window Manager allows users to modify the default behavior to suit their needs. Users can modify the default behavior of OSF/Motif Window Manager by changing the entries in the resource files that it uses to configure its appearance and behavior. Clients also have a number of configuration files.

9.2.3.2.1 Window Types

As client applications and code are written, particular types of windows will be used to fulfill the specific needs of the design plan. The MWM recognizes the following types of client windows:

- Primary - a primary window is a top-level window, a direct descendent of the root window. MWM provides this type of client window with a window frame. By default, this frame is decorated with the full set of functional frame components (resize frame handles, title bar, and window control buttons). The window decoration on primary windows can be changed either programmatically from a client or by using the resource files.
- Secondary - a secondary window is a window that is transient in nature. A secondary window is associated with another window, usually a primary window, and is always over that window in the window stack. Secondary windows are iconified (minimized) together with their associated windows. A secondary window may also receive keyboard or pointer input that it does not pass on to its associated window. This is known as being "modal" with respect to the associated window. A secondary window typically receives less window frame decoration than a primary window, and, typically, fewer window management functions are available to control the window.
- Client Icon - a client icon is supplied by a client for use as an image in an MWM icon.
- Client Icon Window - a client icon window is supplied by a client for use as an alternative to a pixmap image in MWM icons (minimized windows). This window can be used while the window is in its iconic state.
- Override-redirect - An override-redirect window is typically visible for only a short time and, while in use, the pointer should be grabbed by the client. A common example of this type of window is a pop-up menu. MWM does not place override-redirect windows in a window frame, nor does MWM support window management operations on override-redirect windows.

Certain windows constrain the user's input. There are three levels of window constraints, called "modes."

- Modeless windows do not constrain user input to other windows. Client primary windows are generally modeless.
- Application Modal Windows "prevent" input from going to an associated application's windows.
- System Modal Windows are similar to application modal windows except that they prevent input from going to ANY other window on the screen.

MWM uses the following window types to provide window management services to your client application:

- Client Frames - a client frame is placed around the client area.
- Icon Frames - an icon is a small graphic representation of your client application window. When the window manager "minimizes" ("iconifies") a full-sized client window, it uses an icon window frame to represent the client. Icons can be arranged on the screen by the window manager or placed in an icon box.
- Icon Box - An icon box is a window used by the window manager to contain icons. An icon box window is decorated with a window frame that is typically the same as a primary window's frame.
- Feedback Window - A feedback window displays at the center of the screen the size or location of a primary client window which is being either resized or repositioned by the window manager.

9.2.3.2.2 Functions of the MWM Window Frame

The OSF/Motif Window Manager surrounds client windows with a functional frame. Positioning the pointer on a part of the frame and performing the appropriate mouse button action or key action executes the function of that frame part.

The parts of the MWM window manager frame and their functions are:

- Title Area - used to move a window.
- Window menu button - used to display the window menu.
- Minimize button - used to iconify the window.
- Maximize button - used to expand a window to maximum size.
- Resize frame handles - Stretch or shrink a window horizontally, vertically, or diagonally (in two directions).

The window menu contains selections that provide consistent function from one MWM application to another. This consistency reduces the time it takes a user to learn to manage your application windows. The window menu provides an additional way to access window manager functionality. You can select items in the window menu with either the mouse or the keyboard. The available menu selections are:

- Restore - restores a window to its normal size from an icon or after maximizing.
- Move - changes the location of a window.
- Size - changes the width and height of a window.
- Minimize - shrinks a window to its icon (graphic representation).
- Maximize - enlarges a window to its maximum size.
- Lower - places a window at the bottom of the window stack, the position closest to the workspace (root window).
- Close - terminates the client.

9.2.3.2.3 Communicating Between MWM and Clients

A user can set up communications between a client and MWM and configure MWM resources and functions (which might affect your client application). The following topics will help in the establishment of communications:

- MWM Programmatic Interface Standards.
- Inter-Client Communication Conventions.
- MWM Specific Information.

The MWM programmatic interface is based on the Inter-Client Communications Conventions Manual (ICCCM). The ICCCM establishes the standards for "good citizenship" among clients in a multi-client environment. The OSF/Motif toolkit supports the inter-client communication conventions and facilitates appropriate communication with MWM.

9.2.3.3 User Interface Language

The User Interface Language (UIL) is a specification language for describing the initial state of a user interface for a Motif application. The specification describes the objects (for example, menus, form boxes, labels, and push buttons) used in the interface and specifies the functions to be called when the interface changes state as a result of user interaction.

To create a user interface with UIL and the Motif Resource Manager (MRM), the following steps need to be performed:

- Specify the user interface in a UIL module, which is stored in a UIL specification file.
- Compile the UIL specification file to generate a User Interface Definition (UID) file.
- In the application program, use MRM run-time functions to open the UID file and to access the interface definitions. MRM builds the necessary argument lists and calls widget creation functions in the Motif Toolkit.

Using the UIL, the following can be specified:

- Objects (widgets and gadgets) that comprise the interface.
- Arguments (attributes) of the widgets and gadgets desired.
- Callback functions for each object.
- The widget tree for the application.
- Literal values which can be fetched by the application at run time.

The UIL compiler has built-in tables containing information about widgets. For every widget in the Motif Toolkit, the UIL compiler knows the widgets that are valid children of the widget, the widget arguments, and the valid callback reasons for the widget. The UIL compiler uses this information to check the validity of an interface specification at compilation time, to help reduce run-time errors.

The benefits of using the UIL and the MRM are easier coding, earlier error detection, separation of form and function, faster prototype development, and interface customization.

UIL offers the following features to increase productivity and the flexibility of programs:

- Named Values - Instead of directly specifying the values for widget and gadget attributes, named values can be used which are similar to variables in a programming language. A literal value (such as an integer or string) can be given a name and then the

name can be used in place of the value specification. In addition, MRM functions can be used to fetch named values from the UID file for use at run time.

- Compile-Time Expressions - Expressions can be used to specify values in UIL. A valid UIL expression can contain integers, strings, floating-point numbers, Boolean values, named values, and operators. Using expressions can make values more descriptive and can help to avoid recomputing values.
- Identifiers - Identifiers provide a mechanism for referencing values in the UIL that are provided by the application at run time. In the application program, an MRM function is used to associate a value with the identifier name. Unlike a named value, and identifier does not have an associated data type. You can use an identifier as an attribute value or callback procedure tag, regardless of the data type specified in the object or procedure declaration.
- Lists - UIL allows the creation of named lists of attributes, sibling widgets, and callback procedures that can later be referred to by name. This feature allows the reuse of common definitions by simply referencing these definitions by name.
- Support for Compound Strings - Most Motif Toolkit widgets require strings used in the user interface (labels, menu items, and so on) to be compound strings. UIL fully supports the use of compound strings, including left-to-right and right-to-left writing direction and choice of fonts.
- Include Files for Useful Constants - The Motif Toolkit provides a UIL include file that contains useful constants for coding a user interface.

9.2.4 Applicability to the Concept Executive

The OSF/Motif environment is not really an example of an executive, however it is tool which could be used within the scope of a Concept Executive to develop both the executive and the user applications. The OSF/Motif environment would provide the end-users of the developed applications with a behaviorally consistent graphical user interface.

By providing consistent behavior the users' ability to perform a task would be enhanced by enabling them to focus on the task itself rather than on the tools or the methodology they use to perform the task. Motif-developed applications will help to provide this consistency across the system.

Probably the most useful benefit of the Motif system lies in its implementation of Convenience Functions. These functions are callable routines which provide a common interface window which requires a grouping of individual widgets. By making one function call in the application program and providing the appropriate arguments a designer can instantiate a number of widgets. This is extremely useful for interface objects which always require multiple widgets. An example is an error message notification. When an error is detected in the system, a message is usually displayed informing the user of what problem is occurring and a prompt is also displayed for verification. With the convenience function, ErrorDialog, this window can be displayed with minimal coding on the part of the application. Not only do the convenience functions provide simplified application code for creating groups of widgets, but they also help to maintain the consistency of the system by creating the widgets in the same manner.

Another feature of the Motif system which is useful, is the classification of Gadgets. Gadgets are set of widgets which have restricted resource settings and do not have windows, translations, ac-

tions, or popup children. These objects obtain a number of their visual resources from their parent. This results in a much lower requirement for the storage of each widget instance. The main motivation for using gadgets is better execution time and data space performance.

As for the usefulness of the UIL, SwRI could not gather that the UIL was an automated process, such as TAE, which means that the system developer would basically have to learn a new language as well as learn the Motif widget set. It is SwRI's opinion that the programmer would be better off becoming familiar with the Motif widget set and instantiating the objects from within the application code.

10.0 Standards

The chapter will discuss several software standards which are specified as part of the Concept Executive. These standards are presented in detail to allow the reader to help understand why the specific standards were selected. Note that the only standards presented are those which were not specified as part of the Hardware Independent Software Development Environment (HISDE) prototype. This is not to say that the Concept Executive is using different standards, but rather to reflect updates in standards since HISDE was developed. Figure 10-1 compares the standards used in HISDE and in the Concept Executive.

System / Standard	HISDE	Concept Executive
Operating System	System V Version 3	**System V Version 4 POSIX**
Primary Language	Kernigan and Ritchie C	**ANSI C**
Command Line	Bourne shell	**Korn shell POSIX 1003.2**
Networking	International Standards Organization (ISO) OSI Model	International Standards Organization (ISO) OSI Model **POSIX 1003.8**
Real Time	None	**POSIX 1003.4**
Graphic User Interface	X Windows X Toolkit Athena Widgets	X Windows, X Toolkit **Motif Widgets, UIL, mwm P1201**
Graphics	GKS PHIGS	GKS PHIGS

Legend:

- *Standard items listed in* **bold** *text are new standards.*

Figure 10-1 Comparison of Standards

The standards which are described in this document include the following:

- Portable Operating System Interface Definition (IEEE POSIX 1003).
- Windowing Standards (IEEE P1201).

- UNIX System V Release 4 (SVR4).
- ANSI C.

Note that the Motif user interface standard which is specified by the Concept Executive is described fully in Chapter 9.

10.1 POSIX

The Portable Operating System Interface Definition (POSIX) is the standard operating system being defined by the Institute of Electrical and Electronic Engineers (IEEE). POSIX is a standard which specifies all interfaces to the operating system and related support software (networking, real-time functions, etc.). The POSIX effort is significant, as it is the only interface which is globally accepted and is independent of the direction of any single vendor or consortium. This is opposed to offerings from UNIX International (UI) and the Open Software Foundation (OSF) which are driven by certain vendors. The goal of POSIX is to provide application portability across a number of operating systems, including those not based upon UNIX.

POSIX is currently not a fully defined standard. At this time, only the POSIX 1003.1 specification is complete. The 1003.1 standard specifies only the programmatic interfaces to the operating system (the equivalent of UNIX system calls). Completed standards are not yet available for command line, network, real-time or other functional interfaces. These interfaces are being addressed by working groups, each of which is at a different level of completion. Although some working groups are nearing completion, it will be several years before the complete standard is available and implemented on a variety of systems.

It is very important to note that POSIX is not an implementation of an operating system. Rather it is a specification of the programmatic and command-level interfaces. This allows for a wide variety of fundamentally different operating systems to provide a POSIX interface and therefore allow development of portable applications.

The reference model for the POSIX interface is the UNIX operating system. The majority of POSIX interfaces closely match existing UNIX interfaces and behavior. Therefore, the most appropriate operating system selection for environments desiring future POSIX compliance is UNIX.

The following sections discuss in more detail the status and direction of several of the POSIX working groups. The working groups covered are as follows:

- POSIX 1003.2 - Command line interface.
- POSIX 1003.4 - Real-time interface.
- POSIX 1003.6 - Security interface.
- POSIX 1003.8 - Network interface.

10.1.1 Contact Point

The IEEE POSIX Standards are prepared by various working groups, sponsored by the Technical Committee on Operating Systems of the IEEE Computer Society. For information on these working groups contact:

Secretary, IEEE Standards Board,

Institute of Electrical and Electronics Engineering, Inc.

345 East 47th Street

New York, NY 10017

10.1.2 1003.2 - Shell and Application Utility Interface

The POSIX 1003.2 working group was formed to define a standard source code level interface to shell services and common utility program for application programs conforming to POSIX. This group began work in 1985 and is currently in the balloting process for its User Portability Extension. The primary goal for this working group was to specify a standard interface that may be accessed in common by both applications programs and user terminal-controlling programs to provide services of a more complex nature than the primitives provided by POSIX 1003.1. This interface shall be capable of being implemented on conforming systems. It shall include the following components:

- Application program primitives to specify instructions to an implementation-defined "shell" facility.
- A standard command language for a shell that includes program execution, I/O redirection and pipelining, argument handling, variable substitution and expansion, and a series of control constructs similar to other high-level structured programming languages.
- A recommended utility syntax for utility naming and argument specification.
- Primitives to assist applications programs and the shell language in parsing and interpreting utility arguments.
- Recommended environment variables for use by shell scripts and application programs.
- A minimum directory hierarchy required for the shell and applications.
- A group of utilities that may be called from applications programs for complex data manipulation and other tasks common to many applications.
- An optional group of utilities to be used for the software development of applications.
- Utilities and standards for the installation of applications.

The initial focus of 1003.2 was to standardize services via a C language interface. Future revisions are expected to contain bindings for other programming languages as well. This will be accomplished by breaking the standard into two parts, a section defining core requirements independent of any programming language, and a section composed of programming language bindings.

The core requirements section will define a set of required services common to any programming language that can be reasonably expected to form a language binding to this standard. These services will be described in terms of functional requirements and will not define programming language-dependent interfaces. Language bindings will consist of two major parts. One will contain the programming language's standardized interface for accessing the core services defined in the programming language-independent core requirements section of the standard. The other will contain a standardized interface for language-specific services.

The working group consulted a number of documents in the course of its deliberations, to select utilities and features. The three primary reference documents were:

- The System V Interface Definition (SVID), Issue 2, Volume 2.
- The X/Open Portability Guide (XPG), Issue II, Volume 1.
- The ANSI/X3.159-198x Programming Language C Standard.

The current document produced by this working group is broken into nine parts:

- Global Concepts.
- The environment interfaces provided to applications.
- The C language interfaces provided to applications.
- The shell command line interpreter language.
- Descriptions of the utilities in the required "Execution Environment."
- Descriptions of the utilities in the required "Application Installation Environment."
- Descriptions of the utilities in the optional "Software Development Environment."
- Descriptions of the utilities in the optional "C Development Environment Utilities."
- Descriptions of the utilities in the optional "FORTRAN Development Environment Utilities."

10.1.2.1 Global Concepts

The global concepts covered by the 1003.2 working group for the standardization of Shell and Application Utility Interfaces include:

- Stand-alone Utilities.
- Character Set.
- Regular Expression Notation.
- Pattern Matching Notation.
- Utility Conventions.
- Utility Description Defaults.
- File Format Notation.
- Symbolic Limits.

Utilities defined in this standard may be implemented as built-in utilities within the command language interpreter. This is usually done to increase the performance of frequently-used utilities. All implementations conforming to this standard may provide only built-in versions of cd, getopts, and umask; however, the standard shall provide stand-alone versions of all other utilities defined for Execution Environment, Application Installation, Software Development Environment, C Development Environment, and FORTRAN Development Environment. These stand-alone versions are in addition to any built-in versions that may be provided.

10.1.2.2 Environment

Environment variables defined by the standard are of interest to multiple utilities. There are other environment variables that are of interest to specific utilities. Environment variables that are of interest to specific utilities are defined as part of the utility description. The following environmental variables are made available to each utility during execution:

- PATH - the sequence of path prefixes that certain functions apply in searching for an executable file known only by a file name. This variable shall be defined.
- EDITOR - the name of the program the user wishes to employ for editing files. This variable may be defined.
- HOME - the name of the user's home directory, from the user database. This variable may be defined.

- LOGNAME - the name of the user's login account, corresponding to the login name in the user database. This variable may be defined.
- MAIL - This variable may be defined.
- SHELL - This variable may be defined.
- TMPDIR - This variable may be defined.
- USER - This variable may be defined.
- LANG - This variable may be defined, but the format and allowable values are not defined by the standard.
- LC-COLLATE - This variable may be defined, but the format and allowable values are not defined by the standard.
- LC_CTYPE - This variable may be defined, but the format and allowable values are not defined by the standard.
- LC_MONETARY - This variable may be defined, but the format and allowable values are not defined by the standard.
- LC_NUMERIC - This variable may be defined, but the format and allowable values are not defined by the standard.
- LC_TIME - This variable may be defined, but the format and allowable values are not defined by the standard.
- TERM - This variable may be defined, but the format and allowable values are not defined by the standard.
- TZ - This variable may be defined, but the format and allowable values are not defined by the standard.

There are a number of files and a directory hierarchy which are required by the standard. The following directories shall exist on a conforming implementation:

- / (the root directory); applications shall not be allowed to create files in this directory.
- /dev - contains device specific files; applications shall not be allowed to create files in this directory.
- /tmp - made available for programs that need a place to create temporary files; applications may create files in this directory.
- /local and /usr/local - holds local application commands; not required, but if it exists, applications cannot create files in these directories.
- /usr/man - contains reference manual pages; not required, but if it exists, applications cannot create files in these directories.

A conforming system shall contain the following files:

- /dev/null - an infinite data sink; data written here is discarded; reads from here always return zero bytes.
- /dev/tty - a synonym for the control terminal associated with the process group of a process.

10.1.2.3 C Language Interface Option

The C Language Interface option defines interfaces that allow C language applications to access the shell command language and to process regular expressions, command arguments, and expandable file names in a standard manner. The functions which provide these interfaces are:

- Shell Command Interface:
  - system() - execute command.
  - popen(), pclose() - pipe communications with programs.
  - getenv() - access environment variables.
- Regular Expression Parsing:
  - regcomp() - compile the regular expression.
  - regexec() - match a null-terminated string against a compiled regular expression.
  - regfree() - free any memory allocated by regcomp().
- Command Option Parsing:
  - getopt() - command-line parser that returns the next option letter in *argv* that matches a letter in *optstring*.
- File Name Generation:
  - glob() - pathname generator.
  - globfree() - free space associated with glob().
- Get POSIX Configuration Value:
  - posixconf() - return a pointer to a configuration defined value.

10.1.2.4 Shell Command Language

The shell is a command language interpreter. The standard describes the syntax of that command language as it is used by the *sh* utility and the *system()* and *popen()* functions. The working group selected the Bourne Shell as the starting point for its standard. It consciously omitted the BSD C Shell from consideration, for the following reasons:

- Most "portable" shell scripts assume the Bourne Shell.
- The majority of tutorial materials on shell programming assume the Bourne Shell.
- The Bourne Shell is an acknowledged better performer on most implementations.

Despite the selection of Bourne, the working group did not limit the possibilities for a shell command language that was upward-compatible.

Only the system() and popen() function interfaces are described by this standard. Implementation may provide terminal interface programs that allow system users to directly interact with the system using the language described here, and typically provide extra facilities that are suitable for such usage.

The following Korn Shell features are in the 1003.2 Shell Command Language, but not in the System V, Release 3 Bourne Shell:

- Operators - (( )) and < >.
- Reserved words - [[and]].
- Substring expansions - ${*name#pattern*}, ${*name%pattern*}, ${*name##pattern*}, and ${*name%%pattern*}.

- String length expansion - ${#*name*}.
- Command substitution syntax - $(*command*).
- The variable PWD.
- Built-in commands - alias, bg, fg, typeset, and unalias.
- Test operators - -nt, -ot, -ef.
- Assigning values with export and read-only.
- Symbolic names for signals and traps.

The working group has defined the following features to be included in the Shell Command Language:

- Reserved words: case, do, done, elif, else, esac, fi, for, function, if, then, until, while, { }, [[,]].
- Comments.
- Line Continuation.
- Quoting.
- Parameters and Variables.
- Word Expansions.
- Redirection.
- Exit Status for Commands.
- Shell Grammar:
  - Simple Commands.
  - Pipelines.
  - Lists.
  - Compound Commands.
- Signals and Error Handling.
- Shell Execution Environment.
- Job Control.
- BNF for the Shell.
- Special Built-in Commands: alias, bg, colon, dot, break, continue, eval, exec, exit, export, fg, readonly, set, shift, trap, type, set, unalias, unset.

10.1.2.5 Execution Environment Utilities

The Execution Environment Utilities are the utilities that shall be implemented in all conforming 1003.2 systems. These utilities include:

- asa - interpret ASA carriage control characters.
- awk - pattern scanning and processing language.
- basename - return nondirectory portion of pathname.
- bc - arbitrary-precision arithmetic language.
- cat - concatenate and print files.

- cd - change working directory.
- chgrp - change file group ownership.
- chmod - change file modes.
- chown - change file ownership.
- cmp - compare two files.
- colldef - define collation sequence.
- comm - select or reject lines common to two sorted files.
- cp - copy files.
- cut - cut out selected fields of each line of a file.
- date - display the date and time.
- dd - convert and copy a file.
- diff - compare two files.
- dirname - return directory portion of pathname.
- echo - writes its arguments to standard output.
- ed - text editor.
- egrep - search a file for a regular expression.
- env - set environment for command execution.
- expr - evaluate arguments as an expression.
- false - return false value.
- fgrep - search a file for strings.
- find - find files.
- fold - filter for folding lines.
- getopts - parse utility options.
- grep - file pattern searcher.
- id - return user identity.
- hd - hexadecimal dump.
- join - relational database operator.
- kill - terminate or signal processes.
- ln - link files.
- logname - return user's login name.
- lp - send requests to an LP line printer.
- ls - list directory contents.
- mkdir - make directories.
- mkfifo - make FIFO special files.
- mktemp - make a name for a temporary file.
- mv - move files.

- nohup - run a utility immune to hangups and quits.
- paste - merge corresponding or subsequent lines of files.
- pax - portable archive interchange.
- posixconf - get POSIX configuration values.
- posixlog - save messages.
- pr - print files.
- pwd - return working directory name.
- read - read.
- rm - remove directory entries.
- rmdir - remove directories.
- sed - stream editor.
- sendto - send message.
- sh - shell, the standard command language interpreter.
- sleep - suspend execution for an interval.
- sort - sort, merge, or sequence check files.
- stty - set the options for a terminal.
- sum - display file checksums and block counts.
- tee - pipe fitting.
- test - condition evaluation utility.
- touch - change file access and modification times.
- tr - translate characters.
- true - return true value.
- tty - return user's terminal name.
- umask - set file mode creation mask.
- uname - return system name.
- uniq - report or filter out repeated lines in a file.
- uuname - list uux names of known systems.
- uux - execute commands on remote systems.
- wait - await process completion.
- wc - word, line, and byte count.
- xargs - construct argument list(s) and execute command.
- xform - transform collation sequence.

10.1.2.6 Application Installation Utilities

No single installation utility or procedure is sufficient to work with all possible applications, since specialized applications have unique needs. The following provides the minimal requirements for any installation procedure.

An implementation shall provide at least one installation utility. The name and syntax of this utility is implementation-defined. This installation utility shall be part of an installation procedure. This procedure shall be capable of the following:

- Describing the application; for example, listing components.
- Describing the necessary conditions for installation; for example, space required.
- Describing the consequences of installation; for example, files overwritten.
- Installing the application.

10.1.2.7 Software Development Environment Utilities

The Software Development Environment Utilities are the utilities that shall be implemented in all systems that claim conformance to the optional Software Development Environment. These utilities include:

- ar - create and maintain library archives.
- make - maintain, update, and regenerate groups of programs.

10.1.2.8 C Development Environment Utilities

The C Development Environment Utilities are the utilities that shall be implemented in all systems that claim conformance to the optional C Development Environment. These utilities include:

- cc - compile C programs.
- lex - generate programs for simple lexical tasks.
- yacc - yet another compiler compiler.

10.1.2.9 FORTRAN Development Environment Utilities

The FORTRAN Development Environment Utilities are the utilities that shall be implemented in all systems that claim conformance to the optional FORTRAN Development Environment. These utilities include:

- fortran - FORTRAN compiler.

10.1.3 1003.4 - Real-Time Extensions

The POSIX 1003.4 committee is responsible for establishing a standard for real-time extensions for POSIX 1003.1. This effort is required for developing portable real-time POSIX applications. The currently available real-time UNIX environments can be broken into three types of products: host-target approach, rewriting the UNIX kernel, and adding preemption points to the UNIX kernel.

VxWorks, VRTX, and pSOS are examples of the host-target approach to real-time UNIX. This approach, requires at least two processors. One processor runs UNIX while another processor or processors run a proprietary operating system. All real-time functionality is contained in the proprietary system. The UNIX processor usually performs operator I/O or other non-real time operations. Each proprietary operating system will support several different vendors' hardware; however, there is no portability between these operating systems. Porting between supported hardware configurations requires recompiling. This approach has produced systems with interrupt response times of less than 100 microseconds, the fastest of the three approaches.

LynxOS and Regulus are examples of rewriting the UNIX kernel to support real-time applications. One major problem with UNIX in a real time-environment, is the time spent in the kernel. Rewriting the kernel results in interrupt response times in the 200-300 microsecond range. To an application the operating system looks like SVID. Code may be ported from an SVID system with recompilation. But applications written with LynxOS or Regulus real time-extensions will not be portable to any other operating system.

RTU by MASSCOMP and VENIX by VentuCom Inc. are examples of adding preemption points to the UNIX kernel. This approach allows the use of AT&T kernel code, but results in the slowest interrupt response time of the three approaches (guaranteed 3 milliseconds for RTU). RTU only runs on MASSCOMP computers and VENIX runs only on Intel 80286 and 80386 platforms. Thus, applications written for these operating systems are not portable to other systems.

The above approaches do not allow the development of portable applications. POSIX 1003.4 tries to remedy this situation. The major additions specified in POSIX 1003.4 / Draft 8 are: binary semaphores, process memory locking, shared memory, priority scheduling, asynchronous event notification, timers, IPC message passing, synchronized I/O, asynchronous I/O, real time files, and threads. Metrics are included in the specification to aid in evaluating the performance of vendors implementation of the real time extensions. This is important since there is no required performance for compliance. In fact an implementation may keep the AT&T kernel without exemption points and still be POSIX compliant. An implementation may not actually implement any of the above additions and still be compliant since all of the additions are listed as optional in the POSIX standard.

10.1.3.1 Binary Semaphores

Binary semaphores are a high-performance process synchronization mechanism. They are accessed through special files in the filesystem namespace. The binary semaphore is the lowest level of process synchronization.

A binary semaphore may have two states, locked and unlocked. A process may lock a resource from use by any other process. When the process has completed use of the resource, the process unlocks the binary semaphore.

10.1.3.2 Process Memory Locking

Process memory locking allows an application to lock a process in memory. In other words, the process will not be swapped to a mass storage device. This increases throughput since swapping to disk is relatively slow. This also allows a deterministic response time for an event since it prevents the operating system from swapping out the time critical process. Processes that would be memory locked include interrupt handlers and processes that are time critical.

10.1.3.3 Shared Memory

Shared memory is the fastest way for processes to pass information. Shared memory is accessed as a special file. Thus, ***open()***, ***close()***, and other file functions are used on shared memory files. A process establishes a map between the process' virtual address space and the associated special file. Thus, a process accesses the shared memory directly rather than from using ***read()***, ***write()***, and ***lseek()***.

10.1.3.4 Priority Scheduling

Priority scheduling allows the applications to prioritize processes. In a standard UNIX system an aging process is applied. This means that processes are scheduled by the amount of CPU time they have used, not by the time criticalness of their process. Priority scheduling allows processes to take more CPU time if they are time critical.

POSIX 1003.4 specifies two types of scheduling algorithms, first in-first out (FIFO) and round rob-in (RR). FIFO allows a process to run to completion or until the current process yields to another process. RR allows a process to run for a specified interval. When the interval time has expired, the current process is moved to the tail of the process list and will run again when all processes of the same priority have run for one interval.

10.1.3.5 Asynchronous Event Notification

Asynchronous event notification allows a process to keep track of multiple asynchronous events that are performed in parallel. An event includes asynchronous I/O completion, timer expiration, message arrival, and user defined event occurrence.

This mechanism is created to correct deficiencies in the traditional UNIX signal mechanism. First, signals in UNIX could get lost. In POSIX event notifications will not get lost. In traditional UNIX systems, queued signals do not have a defined order of delivery. In POSIX events are delivered in a FIFO order. The relationship between signals and events is implementation defined.

In addition to the two requirements listed above, POSIX events also allow polled event reception, for a high performance notification mechanism, and allow the application to uniquely determine the event. The application may uniquely determine the event that occurred because the event mechanism passes a user supplied value. Prioritized event handling is also supported.

10.1.3.6 Timers

Timers in POSIX 1003.4 allow for nanosecond resolution. Four types of timer types are supported: periodic, offset, relative, and absolute. A periodic timer allows a process to be scheduled at a defined frequency. This type can be used in conjunction with the other three timer types. An offset timer delays a given time between a program level request and the scheduling of a program. A relative timer delays a given time after the recognition of an internal event. And an absolute timer establishes an absolute time, not a time delay, when a program will be scheduled by the operating system.

10.1.3.7 IPC Message Passing

Interprocess communication (IPC) message passing allows processes to send and receive messages to and from message queues. This may be done synchronously or asynchronously. The message queues are system wide and exist in the file system name space as special files.

IPC queues may have multiple writers and multiple readers; however, reading a queue is destructive. IPC queues may be managed using different models. The simplest model is the copy model in which the operating system copies the message buffers before the sender continues execution, or copies the message buffer to a receiver defined location. A move model keeps one copy of the message buffer. This results in the sender losing access to the buffer until the receiver frees the buffer. Another type of model is the use of shared memory to store the buffer. This assumes both sender and receiver have access to the same shared memory area. The sender losses control of the buffer until the receiver frees the buffer.

POSIX 1003.4 has created this new message passing facility due to the inefficiencies of SVID message queues and streams, and BSD's sockets. SVID and BSD facilities did not allow time stamps or out-of-order receipt based on classification of messages. Thus, SVID and BSD applications must be rewritten to support IPC message passing.

10.1.3.8 Synchronous and Asynchronous I/O

POSIX 1003.4 specifies synchronous and asynchronous I/O operations. Synchronized I/O operations are required to complete before program execution is returned to the calling process. This insures that data has transferred to the output device. Their are two levels of data integrity specified. One is data only integrity. The second is file integrity; that is, integrity is assured for the data and the parameters associated with the file. A file synchronization function is included in the specification. This forces all outstanding I/O for a given file descriptor to be completed.

Asynchronous I/O is provided to allow queueing of I/O operations. This allows a process to perform I/O to a single file multiple times or to multiple files multiple times. Completion status may be obtained by polling or by asynchronous event notification, and a process may cancel requests before they are completed.

10.1.3.9 Real Time Files

Real time file specifications allow an application to specify the characteristics regarding how file requests should be handled. At the time a file is created using ***mkfile()***, certain flags may be set to optimize performance for the application. Some of these flags are discussed below.

A contiguous flag indicates the file should be optimized for sequential access. For rotating media this usually means sequential physical blocks to minimize head movement. Two types of flags exist. One indicating that a contiguous file is desirable. The operating system will assign a contiguous file space if it can. The second is a flag that a contiguous file is required. If the operating system can not allocate a contiguous file, the operation fails.

An advisory flag exists for random access. This implies the file will primarily be accessed randomly. Thus, the system will optimize the file for random access if possible. A majority random access flag does not exist.

Flags exist for the use of cache memory. The application may specify the cache be optimized for random or sequential access. There is also a flag indicating the cache will be of no use since the data will not be accessed on a regular basis. The operating system need not use these flags.

10.1.3.10 Threads

A thread is a sequential flow of control within a process. POSIX defines a set of operations to allow for multiple threads in a single process. Threads provide an efficient way for a process to take advantage of systems with parallel architectures. Each thread has its own thread ID, schedule priority and policy, and the required system resources to support a flow of control. A process using multiple threads gives each thread access to most of the state associated with the process. An application that uses multiple processes does not have this sharing of state. Also, the creation and removal of a thread should be faster than creating and removing processes.

The thread section introduces the concept of mutexs. Mutex is derived from mutual exclusion, and this is the capability it performs. Mutexs are similar to binary semaphores except they are defined to work between threads within a process, not between processes. Another difference is the thread

that locks the mutex is the only thread that may unlock it. This is not true of the process that locks a binary semaphore.

Mutexs work in conjunction with condition variables. If a thread holding a mutex determines it can not proceed by examining data guarded by the mutex, it loops on a condition variable. Other threads may alter the condition variable to allow the thread to continue execution.

According to Draft 8 used for this summary, mutex functionality may be added to the binary semaphore proposal.

10.1.4 1003.6 - Security

The POSIX 1003.6 standard defines an interface for security functions in a portable operating system. The scope of this interface includes the definition of new system functions and commands for the additional security mechanisms supported by the standard, as well as new, security-related constraints and requirements for the functions and commands defined by the other POSIX standards. The P1003.6 security subjects and objects are identified, but no explicit policy model for their behavior has been defined yet. The P1003.6 security policies are defined precisely, however, and the correspondence between the interfaces and policy is demonstrated. The requirements for trusted systems are drawn primarily from the Trusted Computer System Evaluation Criteria (TCSEC). The committee will consider other requirements of reasonable generality for inclusion into the standard and will allow extensions that are consistent with the standard. Since the purpose of the security standard is to provide for application portability between trusted systems, security requirements in areas which do not pertain to application portability are not addressed. Functions and commands which relate to Mandatory Access Control (MAC) are identified as part of a separate option, which need not be provided or configured on all Conforming Systems.

Although the P1003.6 standard does not mandate a specific security policy model, it does define functions and commands which may be used to support a number of types of security policies. These types include:

- Nondisclosure - defines constraints upon the observation of information. Access control mechanisms are provided which allow for discretionary and mandatory constraints on the reading of an object by a subject.
- Integrity - define constraints upon the alteration of information. Access control mechanisms are provided which allow for discretionary constraints on the writing of an object by a subject.
- Accountability - Accountability policies define which user actions are auditable. Auditing mechanisms are provided for the collection and analysis of security-relevant events on a per user basis.

The P1003.6 committee has just begun to define the actual contents of the security standard. At this point, the standard defines the functions and commands which will support:

- Discretionary Access Control.
- Mandatory Access Control.
- Security Auditing.
- Least Privilege.
- Object Reuse.

- Object Import and Export.

Discretionary Access Control is provided by functions which support Access Control Lists on the appropriate POSIX objects. Access Control Lists allow users to assign control attributes associated with a security object or a group of security objects, an object being a passive entity that contains or receives information. Access to an object potentially implies access to the information that it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees, and programs, as well, as bits, bytes, words, fields, processors, video displays, keyboards, clocks, printers, network nodes, etc. The control attributes which may be specified are attributes associated with an object that, when matched against the security attributes of a subject, are used to grant or deny access to that object.

Mandatory Access Control is provided by functions which support the labeling of subjects and objects, and by mandatory access control rules sufficient to enforce the TCSEC information confinement policies. Additionally, all interfaces are constrained to avoid covert channels.

The requirements for trusted object reuse will be constraints upon those POSIX functions that allocate new objects or extend existing ones, rather than separate system functions and commands.

Trusted object import and export is provided by the inclusion of the security attributes in the file headers defined for the POSIX Archive and Interchange utility (PAX), and by the optional inclusion of file header origination and authentication fields.

Auditing is provided by the definition of auditable events with regard to the P1003.6 interfaces, and by a standard audit trail record format and functions to write records to and read records from the system audit trail.

Least privilege is supported by the definition of the privileges required for POSIX functions limited to processes with appropriate privileges, and by mechanisms to associate these privileges with processes.

10.1.4.1 Discretionary Access Control

The primary goal of POSIX in extending discretionary access control in the UNIX system is to provide a finer granularity of control in specifying user and/or group access to objects. The POSIX approach to achieving this is through the addition of Access Control Lists (ACLs). Discretionary Access Control (DAC) is a means for controlling access to an object based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that they are chosen by the object owner.

The DAC mechanism employed in the current UNIX System was designed for efficiency, flexibility, and ease of use. This mechanism allows and encourages the sharing of information, but at a very coarse granularity, via the use of file permission bits. File permission bits are associated with three classes: owner, group, and other. Access for each class is represented by a three-bit field allowing for read, write, and execute permissions.

Although several methods exist for allowing discretionary access control on objects, ACLs were chosen to provide a DAC mechanism with finer granularity than the current file permission bits. It has been determined that the current DAC mechanism in the UNIX System is adequate for most needs, and that the only enhancement required is to allow reasonable finer-grained control of objects. This provides the capability to allow or deny access to individually specified users and/or groups and meets the B3-level requirements of the Department of Defense TCSEC.

10.1.4.2 Mandatory Access Control

The overall goals of P1003.6 Mandatory Access Control (MAC) are:

- Address disclosure-based mandatory access controls to the POSIX interface standards that fulfill appropriate, widely-recognized standards and criteria as defined in the overall scope for P1003.6. This will include the TCSEC.
- Define MAC interfaces for portable trusted applications, and specify MAC restrictions on all other P1003 functions.
- Apply MAC to fulfill the appropriate standards and guidelines, yet provide as much flexibility for implementation-specific MAC policies as is practical.
- Preserve the provision for POSIX conforming applications to impose "extended security controls" as defined in P1003.1.
- Address MAC-related aspects for all forms of information access and transmission visible via the POSIX interface, including regular data channels (e.g., files) as well as covert channels.
- Preserve full compatibility to base P1003 functionality among subjects and objects operating under "single-label conditions."
- Add no new MAC-specific error messages to existing P1003.1 (and other) interface standards.
- Define interfaces and formats for the interchange of MAC labelled information between conforming implementations. Define various standard MAC label format information as appropriate.

An issue being considered for the MAC area of P1003.6 is that common features of UNIX systems not in POSIX be included in the scope of MAC, for example, System V shared memory, semaphores, and message queues, or BSD sockets.

10.1.4.3 Audit

There are two principal auditing goals for P1003.6. They are portable audit post-processing applications and audit control for trusted applications. A third goal, implied by the other two, is that the interfaces and definitions described in this standard be efficient enough to encourage widespread implementation and use.

In order to provide portable audit post-processing, the standard needs to:

- Define standard portable audit trail formats that allow for implementation-defined events and event information.
- Define a set of events that a conforming implementation must be capable of generating. This includes both the conditions under which the events must be capable of being generated and the information to be included in those events.
- Define standard portable audit trail formats that allow for interoperability, so that audit data can be portable among different machines, and so that audit data generation and analysis need not take place on the same machine.

In order to provide for audit control for trusted applications the standard will:

- Define functions for generating audit records both by applications that are part of the Trusted Computing Base (TCB) and by applications that are outside the TCB. Non-

TCB applications are constrained to use interfaces that are guaranteed to provide correct subject identification and token format in the audit record. A privilege may be required for non-TCB applications to generate audit records (to be defined by the privilege group).

- Define functions that allow the application to disable and enable events associated with the actions of the application that are being recorded by the TCB.

The scope for security auditing will not include:

- Administration - functions and commands to support security audit administration are excluded. These exclusions include the assignment of audit control parameters to specific users, and pre-selection by object.
- Audit control to detailed level - Functions that allow trusted applications to control pre-selection to the detail of specific events or classes of events are excluded. However, global control of pre-selection is included.
- Post-processing classes - the definition of a set of standard post-processing classes is excluded.
- Audit data storage - the definition of formats and organization for permanent audit data storage are not addressed, nor is there any required storage organization for a system's audit trail.
- Audit delivery mechanism - the definition of a mechanism for delivering records to a system's audit trail is not addressed, although the interface to this mechanism is included.

Those functions which will be included in P1003.6 Auditing Functional Specifications can be grouped as follows:

- Audit Record and Token Definitions - A standard audit trail format and conventions for first user are defined.
- Audit Record and Token Interfaces - Standard functions for examining and creating audit records are defined, as well as functions for examining and modifying individual tokens within audit records.
- Audit Control Functions
- Event Definition, Identification, and Content - Conforming implementations are required to be capable of generating specific audit events. These events, their identity and contents, and the conditions under which they are to be generated are defined by invocations of the audit macro in the Common DTLS.

10.1.4.4 Privilege

The P1003.6 Security Interface standard is currently defined to provide a number of commands and functions for privilege protection. These include:

- Provide a common terminology for addressing the topic of privilege.
- Define the semantics of how process privileges are acquired and altered.
- Provide compatibility with existing *setuid()*, for future implementations.
- Define the system functions and commands necessary to support the development and execution of trusted programs, consistent with the principle of "least privilege."

- Provide and solicit input to and from other groups regarding areas of common interest.

10.1.5 1003.8 - Networking

The POSIX 1003.8 Committee is addressing network standards to be included in POSIX. There are four groups working on this effort. One group is working on ISO compliance. The other three groups are working on areas not covered by the ISO standards.

The ISO group is developing programmatic interfaces which are not specified in the ISO specifications. The underlying functionality of the POSIX interfaces provide ISO compliance. A problem the POSIX group will have is that manufacturers have already developed their own ISO applications. Each will be trying to push the standard to their interfaces. Thus, the finalization of this specification is still a long way from being complete.

One standard that all POSIX networking groups will adhere to is the OSI seven layer model. Advantages to this model are the ability to interchange layers in conforming networks. This allows new technology to be inserted with minimal impact on the network. Also, a cost benefit is expected due to faster system implementation, easier maintenance, and competition between vendors of similar products. One other development is the consideration of the transport layer interface of SVR4. The purpose of this would be to provide a standard interface to the lower layers of the seven layer model. Then applications could be developed that will work with TCP/IP networks or ISO compliant networks. This could help the transition from TCP/IP based networks to ISO compliant networks.

The first layer of the OSI model is the physical layer. This layer converts the physical interface between devices and the rules for the data transfer. Several standards currently exist for this layer. The most used are:

- CSMA/CD - Ethernet.
- Token Bus.
- Token Ring.

Several other standards will be adopted in the future. The most important of which is Fiber Distributed Data Interface (FDDI). This will allow high speed (100M bits per second) fiber optic backbone networks.

The data link layer is the second layer of the OSI model and provides packaging of data for transmission. Error detection is provided as well as the ability to activate, maintain, and deactivate the link. The standard for this layer is IEEE 802.2. This standard has been adopted by ISO and GOSIP and should be the POSIX layer two standard.

The third layer, the network layer, determines the data's route. It relieves the transport layer from knowing anything about the routing required for the data to reach its destination. The connectionless protocols have been emphasized in this layer by GOSIP. POSIX may endorse both connection based and connectionless protocols or just the connectionless protocol. A connectionless protocol provides the most general data routing. With a connectionless network layer, all higher layers may specify connection-mode protocols which will work without a direct network connection.

The transport layer, layer four, provides a reliable mechanism for data exchange between different systems. Data integrity is maintained in the transport layer. POSIX has specified Transport Protocol Class 4 (TP4) as the standard for this layer. This provides a connection based method of data transfer.

The session layer, layer five, establishes and synchronizes the communication sessions between applications. It establishes and maintains the connection, called a session, between applications. ISO has specified a session protocol. The use of the protocols depends on the applications in use.

The presentation layer, layer six, provides the translation required for the data format to be understood by both applications. Its purpose is to resolve differences in data format. POSIX specifies the ISO presentation protocol.

The application layer, layer seven, provides the interface between the user application and the network. This allows the application developer an interface to the entire network without knowing the details of the lower layers. Several standards have been created at this layer by ISO. It is assumed that POSIX will address each of these standards as they are released. These include:

- Message Handling System (MHS).
- File Transfer, Access, and Management (FTAM).
- Associate Control Service Element (ASCE).
- Directory Services (DS).
- Virtual Terminal (VT).
- Network Management (CMIP).
- Job Transfer and Manipulation (JTM).
- Office Document Architecture/Office Document Interchange (ODA/ODI).

Below is a description of each protocol.

10.1.5.1 Message Handling System

Message handling system (X.400) is a mature protocol that has been implemented around the world. The current version of the standard (1988) is over 2200 pages. X.400 standardizes four major protocols for interconnecting components of a distributed message handling system. P1 specifies how two computers transfer messages to each other. P2 is a format specification for messages representing office memos. It allows for the submission, transfer, delivery, and retrieval of messages. P3 specifies how one computer submits or takes delivery of messages from a computer responsible for message transfer. P7 specifies how one computer submits or takes delivery of messages from a computer responsible for message storage.

10.1.5.2 File Transfer, Access, and Management

File Transfer, Access, and Management (FTAM) protocol allows applications to transfer files across the network. The basic services provided by FTAM include:

- Reading a remote file.
- Write to an existing remote file .
- Write to a new remote file.
- Append an existing remote file.
- Read a record of a remote file.
- Write a record of a remote file.

FTAM uses a virtual filestore to allow uniform access to files. The virtual file store provides a uniform method of accessing the real file store. The FTAM implementation is required to determine

a mapping between the virtual file store and the real file store. All applications access the virtual file store for remote access to the real file store. Each real file store on the network will have a virtual file store that allows remote access in a uniform manner.

10.1.5.3 Association Control Service Element

Association Control Service Element (ASCE) provides basic facilities for the control of an application association between two applications that communicate with a presentation connection. This control is limited to establishing, releasing, and aborting the association. Other applications such as FTAM use ASCE for these services. Below is a summary of these services:

- A-Association - allows an application to establish an association with another application entity. The two entities may identify themselves by a title and they may identify themselves by specifying an application context. This gives each application an indication of the type of connection and allows negotiation of context if required.
- A-Release - allows for the orderly, negotiated release of the association.
- A-Abort - allows for an application entity to abort the association without negotiating the release. There may be loss of data and the application entities are not informed of the release.
- A-P-Abort - allows for an application entity to abort the association without negotiating the release. The application entities are notified that the association was released. In this service data may be lost.

10.1.5.4 Directory Services

Directory services (X.500) allow the use of logical names in place of physical names and allow alteration to the network without affecting network operation. X.500 provides these services to all entities on the network. Thus, FTAM and X.400 will use the same directory services. Any configuration changes in the network will be hidden from these applications.

X.500 provides three basic services:

- Name to name binding service - allows the binding of logical names to their physical address. This is similar to a white pages listing.
- Name to list of names binding service - allows the binding of one name to multiple names. This service is useful for routine of electronic mail and can be considered a mailing list.
- Property to set of names binding service - allows a list generated by a given property. This can be thought of as a yellow pages service.

10.1.5.5 Virtual Terminal

Virtual Terminal (VT) provides terminal emulation services. This is done by mapping real terminal facilities to abstract terminal facilities. Thus far, basic character based mappings have been addressed, but bit-mapped facilities are being developed. VT also provides network wide terminal emulation. This allows a VT-100 terminal attached to a VAX to use applications on an IBM that assumes the terminal is a 3270. This is done at the time of establishing the terminal connection. Each time a connection is established, several parameters are negotiated. Since the number of parameters may be large, profiles of the most popular terminals may be an important negotiation tool.

10.1.5.6 Network Management

Network management is divided into Common Management Information Protocol (CMIP) and Common Management Information Services (CMIS). They are not an ISO standard yet since they are in draft modes. POSIX will probably not address network management until these documents become standards.

10.1.5.7 Job Transfer and Manipulation

Job Transfer and Manipulation (JTM) is designed to support computer to computer communications for performing work remotely. JTM is optimized for background or batch processing. Loading analysis and optimization is not provided by JTM.

10.1.5.8 Office Document Architecture/Office Document Interchange Format

Office Document Architecture/Office Document Interchange Format (ODA/ODIF) addresses the problem of document interchange. A ISO standard file format (ODIF) will allow vendors to directly support one file format or develop one translation between their file format and the ODIF. This will provide a common method of exchanging document files between different applications.

10.2 P1201 - Windowing Standards

With the widespread endorsement of X Windows many groups have formed to promote the development and standardization of X. Though the X protocol was proceeding toward standardization, the industry was not getting any closer to solving the problems of application and user portability. The few windowed applications available had different user interfaces, and most vendors showed no signs of committing to development of applications for windowed environments. Thus, the IEEE Computer Society Technical Committee on Operating Systems (TCOS) approved the formation of the P1201 working group. The group was chartered to develop standards that would further application and user portability in the X Windows environment.

Since this effort is based on the X Window System model of windowing, a brief description of X is needed to indicate the need for standardization. The X Window System is a network window system based on a client-server model, under which clients and servers can reside on computers having very different architectures. X servers can service several clients simultaneously and, under this arrangement, each client can control one or more windows on the server's display screen.

There are three essential components of the X Window System: clients, servers, and the X Window protocol. An X Window server is responsible for providing windowing services to clients. The X Window protocol, at the very base of the system, is used to communicate between servers and clients, which often run on different nodes of a LAN. Several different clients running on different nodes of a LAN can all use a single display server to interact with a user.

X Window clients are applications which use the X Window System to interact with the user. X Window clients typically consist of several layers of software. Above the protocol layer sits the API, commonly referred to as XLib, which provides a functional interface to the protocol. These routines establish network connections with servers, compose protocol packets, transmit them to the server, and receive replies.

The next layer of routines above Xlib provides routines which manage higher-level user interface objects called *widgets*. A widget provides a specific user interface service. It has a distinctive visual appearance, and a well-defined response to user input. The X Intrinsics layer provides functions which allow an application to create, modify and destroy widgets.

The layer above the Xt Intrinsics is the toolkit layer, which is implemented using a collection of widgets. This layer consists of a set of predefined widgets, which may be used by programmers in building applications. Application writers can pick and choose from the array of widgets that a toolkit offers, or construct their own custom widgets by directly using the X Intrinsics routines. Several different widget sets are available to be used (Athena, HP , Open Look, and Motif). By definition, the widget layer not only provides an API, but also presents a certain look and feel to users.

A very important component of an application, though not a software layer, is the style guide. The style guide describes the overall "look and feel" of an application that conforms to the guide. This collection of widget looks and feels, along with more general interface characteristics, such as the overall composition of the screen and the presence and placement of menu bars and icons, loosely makes up the "style guide." Typical components of "look and feel" are:

- Overall visual appearance of the screen.
- Appearance and positioning of icons.
- Composition and layout of dialog boxes.
- Visual appearance of widgets.
- Opening and closing windows.

Another component of the style guide describes the feel or drivability of an application at a lower level. It describes how each type of widget interacts with the user. Another way to describe drivability is the set of user interface actions used to interact with widgets.

Two more layers which can be added above the toolkit layer are a user interface service layer and a user-interface language layer. The user interface service layer provides services that are independent of both the display and the look and feel; clearly it is not constrained by a particular window system or a particular widget toolkit. Rather than referring to the display of a checkbox at a particular location on the screen, requests to this layer ask for the presentation of a binary-choice service to the user and expect the final state of the choice to be returned. It allows developers to program to a single, high-level service API that can run on the huge base of character-mapped displays and on bitmapped screens.

The user-interface language layer provides programmers with the ability to construct Fourth Generation Language (4GL)-like user interfaces with the same productivity benefits attributable to 4GLs. While there is considerable debate in the industry about where it belongs, this service is nonetheless useful. At the very least, a user-interface language specifies the contents and layout of user screens and dialogs. It may also include a procedural description of possible responses to various user events.

Two objectives that P1201 is focusing on are the development of an Application Programming Interface (API) standard for the toolkit layer of X and the development of a recommended practice for drivability for user interfaces. A standard for drivability is important and useful because it contributes to user portability, by allowing users to move easily between applications and systems from different vendors. The IEEE P1201 committee currently consists of two working groups: P1201.1 and P1201.2. P1201.1 is attempting to create a standard X Window System toolkit interface while P1201.2 is looking at human factors issues relating to user portability across multiple "look and feels."

10.2.1 Contact Point

The P1201 standard is sponsored by the Technical Committee on Operating Systems of the IEEE Computer Society. For information, contact:

Secretary, IEEE Standards Board,

Institute of Electrical and Electronics Engineering, Inc.

345 East 47th Street

New York, NY 10017

10.2.2 Application Programming Interface for Program Portability

Windowed applications need high level services such as buttons, menus, scroll bars and edit fields which are provided by user interface toolkits in the X Window System model. These common services require specific procedural interfaces to programming languages so that applications that use these services can be portable from one system to another. The development of such procedural interfaces for application portability will be part of the work program for the standards group (P1201.1) looking at Application Programming Interfaces (APIs).

Since P1201.1 was formed, both OSF and AT&T have developed their own toolkits, Motif and Open Look. They have different APIs and implement different look-and-feel qualities. Much of the work of the P1201 API group thus far has been devoted to deciding whether the working draft of the API standard should be based on Open Look or Motif, or whether a combined approach should be taken. The question is still open.

One solution to providing a common API across multiple toolkits is the adoption of a user interface management system (UIMS). UIMSs are intended to encourage the separation of a software system into an application portion and a user interface portion. The application portion implements the "core" functionality of a system, while the user interface portion implements the user interface dialogue.

In a summary of the August 1989 meeting of the P1201 committee there was a report that significant discussions were held concerning the use of user interface definition languages (UIDL)/higher level Application Dialogue Interfaces (ADI) as a means of providing a standard API. As a result of this discussion, a study group was chartered to look into the possibility of defining a standard UIMS. In order to do this, the group defined the following process:

- Define the domain of the working group.
- Approve a standard reference model.
- Develop a set of requirements.
- Solicit technologies that address the requirements.
- Evaluate submissions and determine if any are adequate (with or without modification) as a basis for a standard.

The long-term goal of the group is to create either a standard or recommended practice in the area of user interface management systems. It is hoped that this effort will result in a standard API which will remain constant across changes in user interfaces and user interface toolkits. The group has currently completed the first two steps and has developed an initial set of requirements.

The following is the standard reference model for the study group:

- Layer 6 Application

- Layer 5 Dialogue
- Layer 4 Presentation
- Layer 3 Toolkit

This reference model is based on the standard Seeheim UIMS architecture. The numbering scheme is used to correspond to the reference model used by the remainder of the P1201 group. The domain of the study group is currently defined to be:

- Define a standard application dialogue interface (between layers 5 and 6 in the reference model).
- Define a standard UIDL (layers 4 and 5 in the reference model).
- Define a standard binding mechanism for integrating toolkit level API's into the presentation layer (between layers 3 and 4 in the reference model).

The initial requirements for a standard UIMS as specified by the study group are as follows:

- Support a mixed control model. A mixed control model is one in which either the user interface or the application can be in control. An Automatic Teller Machine is an example of an application that is user interface controlled; a radar display is an example of an application that is application controlled. A radar system that allows the operator to modify parameters supports a mixed control model.
- Allow the development of applications which are independent of presentation or media concerns. For example, a single application developed to the ADI must be portable across varying X Toolkit implementations, windowing systems, and I/O technologies (e.g., voice, text, and graphics).
- Be programming language independent. This may be accomplished by providing different language bindings (e.g., C, Ada, or FORTRAN).
- Support runtime requests from the application for the presentation of information to the end user. This is necessary to support applications such as drawing tools, CAD/CAM systems, and other interactive editors. The assumption is that the system designer can not know apriori how many instances of particular object will need to be displayed at runtime.

The UIDL must:

- Be powerful enough to support the definition of the presentation aspects of the interface and dialogue behavior.
- Be able to describe dynamic behavior and in particular must be able to display and remove toolkit objects without application involvement. This is meant to support user interface functionality such as cascading menus and direct manipulation.
- Be able to service dynamic requests for presentation services made by the application.
- Be toolkit independent; the language must not be tied to a single toolkit.
- Provide the ability to manipulate the toolkit objects defined in an arbitrary toolkit.
- Support the manipulation of generic toolkit objects which can then be linked to specific objects in different toolkits.

The toolkit binding mechanism must:

- Be able to support multiple window systems (e.g., X, NeWS, PM, Mac).
- Be able to support multiple toolkits for a given window system (e.g., 1201.1, Motif, OpenLook).
- Be able to support character based terminals (e.g., curses on a VT220).
- Be compatible with existing toolkit interfaces. This means that toolkits can be integrated into the UIMS without requiring modification to the toolkit API.

The UIMS study group is currently soliciting technologies which address the initial set of requirements. To date, the group has heard presentations on OSF UIL; DEC UIL; the Extensible Virtual Toolkit developed by API Ltd.; the Serpent UIMS developed at the Software Engineering Institute at Carnegie Mellon University; Ness, ADEW, and the Andrew Toolkit developed by the Information Technology Center at CMU; and the TeleUSE UIMS developed at TeleLOGIC.

10.2.3 Drivability/Look and Feel

The efforts of the P1201.2 working group will also be focused on user portability through a specification for application drivability. Drivability can be characterized as the set of standard user actions used to invoke application capabilities. The drivability concept applies to a subset of a user interface's look and feel, the screen icons. Consider a push-button, a common feature in almost every Macintosh-like dialog box. As long as the push-button looks something like a button, users will think of it as such, regardless of whether it has rounded edges or square, shadows behind it or a pseudo-3D appearance. Of course, if someone made a button to look like a scroll-bar, there would be a problem. But generally speaking, how the button works is much more important and much less obvious to users than is the button's appearance. A consistent convention for function is required so that toolkit writers have some guidance on how to program commonly used widgets, such as buttons, scroll-bars, and edit fields, so that they respond to user actions in a similar and perhaps standard manner. This convention is the focus of the drivability effort within P1201.2. It can also be described as part of the feel of the application at a 'micro' level. Drivability is not concerned with the appearance of the screen or how icons are opened into windows; it is concerned with the user actions which are used to interact with toolkit widgets displayed on the screen.

Drivability is called out separately from look and feel, because a standard for application drivability is the key to greater user portability. The concept of drivability is important in the network-based X Window system because a user may be presented with windows from different applications and different systems simultaneously. While there is much room for creative variations between applications, the use of a set of common user interface actions enhances user portability and decreases the chance of user error. If each of these systems uses a different standard for drivability, the user would be very confused and would be quite likely to make a serious error. If however, these applications conform to a standard for drivability, a user can switch between them with ease. The two areas of application and user portability are linked together because the design and implementation of a user interface toolkit needs to reflect standard drivability considerations.

10.3 UNIX System V Release 4

System V Release 4 (SVR4) is the most recent release of System V (SVID) UNIX. This operating system was developed by ATT and Sun Microsystems and will be generally available in the first quarter of 1990. SVR4 is the UNIX operating system defined as the "standard" by the UNIX International (UI) Consortium. SVR4 is a merge of the best (in the interpretation of UI) features of

System V Release 3, Berkeley Software Distribution (BSD) 4.3, and Xenix. SVR4 also includes functionality from Sun's SunOS operating system.

SVR4 will compete for market share with the OSF/1 operating system being developed by the Open Software Foundation (OSF). OSF/1 is primarily based on the Advanced Interactive Executive (AIX) operating system developed by IBM. OSF/1 will also use features from Mach for multiprocessor support. Mach is an operating system designed at Carnegie Mellon University.

Rather than comparing the two operating systems, this discussion will describe the advantages and features of SVR4.

10.3.1 Contact Point

Implementations for System V Release 4 are available or planned for a number of computer systems. For information on the standard itself, contact:

UNIX International

20 Waterview Boulevard

Parsippany, NJ 07054

10.3.2 System Description

In the past, UNIX systems were somewhat non-standard due to the differences in the System V, BSD, and other variants. SVR4 promises to eliminate this problem by merging the best features of the three most popular UNIX variants. A summary of the major advantages of SVR4 are as follows:

- SVR4 offers a combination of the best features of the three operating system variants. SVR4 also includes a number of features taken from Sun's operating system (SunOS).
- Applications developed in System V, BSD, and Xenix will be fully compatible with SVR4.
- SVR4 is POSIX-compliant and will remain so as POSIX matures.
- SVR4 offers real-time capability.
- SVR4 is available on selected systems at this time. SVR4 will be generally available on other systems during 1990.
- SVR4 is based on robust software technology and initial releases should be quite reliable.

By combining the best features of System V, BSD, and Xenix and adding new features from SunOS, SVR4 provides an operating system which is compatible with applications developed in any of these environments. This also allows SVR4 to provide a rich environment for development of new applications. The remainder of this section will describe the combinations of features which make up SVR4.

SVR4 includes and is primarily based upon features of System V Release 3.2. The provided features include the following:

- Classic System V interprocess communications facilities such as shared memory, message queues, and semaphores.

- Streams - the System V mechanism for local and network process to process communication. Note that streams will be used as the basis for the implementation of Berkeley sockets.
- UNIX to UNIX Copy Program (uucp) - the System V mechanism for serial-based network communications.
- Transport Level Interface (TLI) - a programmatic interface which allows applications to be independent of the underlying network transport protocol (such as TCP/IP or ISO/OSI). This interface will be very useful for environments making the transition from TCP/IP to ISO/OSI.
- Remote File System (RFS) - the System V network-based file system. RFS is similar to the more familiar NFS, but is specifically geared to support of UNIX filesystems.

SVR4 also includes several useful features taken from BSD 4.3. The provided features include the following:

- The C shell - the familiar shell favored by most users for interactive use.
- The Berkeley fast file system - a file system which improves file access performance via cylinder groups, larger block sizes, and use of a mechanism which maintains contiguous files.
- Selected commands and system calls (including Berkeley signals).
- Symbolic links - which allow links to directories and to files/directories located on different file systems.
- Sockets - the Berkeley mechanism for local and network process to process communication. Note that the socket interface will be implemented with streams. This will have a slight negative impact on performance.
- Remote (r*) commands - remote operation commands such as rcp, rdate, rlogin, rsh, and others.
- The Internet services daemon (inetd) - which provides control of daemon processes which access the Internet.

SVR4 includes a number of significant features of the SunOS operating system. The features include the following:

- Virtual memory mapping mechanism - supports memory mapped files, dynamic linking (linking at run time), and shared libraries. Dynamic linking and shared libraries are particularly useful as they:
  - Allow libraries to be changed without affecting the calling application.
  - Allow all executing applications to share one resident copy of a library.
  - Significantly reduce disk and memory requirements. This is particularly important when using complex software such as X Windows and ISO due to multiple layers of large libraries. Shared libraries also aid Reduced Instruction Set Computer (RISC) machines due to the increased size of RISC executables.
- Network File System (NFS) - supports UNIX and non-UNIX network-based file systems. NFS is useful for sharing file systems and for support of diskless workstations.
- Remote Procedure Calls (RPC) - provides an OSI session layer interface for execution of procedures (functions) on remote workstations.

- EXternal Data Representation (XDR) - provides an OSI presentation level interface for exchanging data between machines with different architecures.

SVR4 includes a number of additional features which are taken from different operating systems or are entirely new. These features include the following:

- Enhanced stream support - the performance of streams has been improved.
- Internationalization - multi-national language support.
- Virtual File System - an additional network file system taken from the Apollo operating system.
- Korn shell - a new command interpreter which is a superset of the Bourne shell. The Korn shell is completely compatible with the Bourne shell for both interaction and execution of shell scripts. The Korn shell includes many of the useful interactive features of the C shell and other new improvements as well.
- New mechanisms/interfaces for network selection, name to address mapping, and mail.

10.3.3 SVR4 Future Directions

UNIX International has provided a well-defined set of directions (called a road map) for introduction of new functionality into SVR4. This road map outlines the addition of support for enhanced security, multiprocessing, and increased network support. Figure 10-2 summarizes these enhancements and how they affect different areas of SVR4.

SVR4 Release / Function	System V Release 4	SVR4 Enhanced Security	SVR4 Multiprocessing Plus	SVR4 Network Computing Plus
Internationalization	Multi-National Language Support	Added Multi-Language and GUI Support	Enhanced Multi-Byte Character Support	
Systems Administration	Enhanced Local Administration	Remote Administration		Distributed Resource Management and Full Function Net Management
Multiprocessing			Multi-Processing Kernel, Parallel Processing API	Distributed Multi-Processing
Network Computing	TCP/IP, NFS, RPC			Full Distributed Applications Support, OSI Support
Security	Security Ready	Certified B2 Level Security		
User Interface Functions	Graphics-based X Windows		Common API for Multiple GUIs	Object Management
Core Operating System	Virtual Memory, VFS, Real-Time Capability		File Management and Transaction Processing Enhancements	Advanced Real-Time

Figure 10-2 SVR4 Future Directions

The timeframe for the different enhancements is not known at this time. It is unfortunate that SVR4 will concentrate on security in preference over multiprocessing, as multiprocessing support is very useful for real-time environments.

10.4 ANSI C

ANSI C has been selected as the C standard primarily for two reasons, portability and industry acceptance. Programs written in ANSI C will have an easier port than programs written in Kernigan and Ritchie (K&R) C. The industry has widely accepted ANSI C as the C standard. This is being demonstrated by the delivery of compilers based on the ANSI C draft standard.

10.4.1 Contact Point

X3 Secretariat/CBEMA

311 First Street, N.W. Suite 500

Washington, DC 20001

10.4.2 Status of ANSI C

ANSI C has not been formally accepted at the time of this document. All technical details have been resolved, but procedural delays have kept the standard from being released.

10.4.3 Objectives of ANSI C

The developers of ANSI C have three objectives in their development. First is compatibility with K&R C. Second is to include common language extensions in ANSI C. Finally, the committee is concerned about portability.

10.4.4 Characteristics of ANSI C

This section briefly describes the characteristics of ANSI C. The major characteristic of ANSI C is to provide a standard set of headers. This greatly increases the portability of code generated in ANSI C. One other characteristic of ANSI C is the addition of function prototypes. This helps in preventing errors in passing parameters in ANSI C code. Several other additions and changes have been made to K&R C. They are summarized below.

10.4.4.1 Standardization of Libraries

A large effort of the ANSI C committee is to generate a set of library standards. A set of 15 headers are included in the standard. This is a major improvement for portability of applications. K&R did not specify headers and header files varied between manufacturers. The ANSI committee resolved these differences into one standard that is not controlled by a manufacturer. This alone is justification for adopting ANSI C.

10.4.4.2 Function Prototypes

An important addition to the language is function prototypes. This allows the compiler to check parameter types for consistency. Function prototypes are not required which allows K&R programs to compile with ANSI C compilers. A typical function prototype is:

```
function ( char *, int, unsigned int * ).
```

This forces any calls to function to pass the required number of parameters and the proper parameter types. An alternate method is to write the function prototype as follows:

```
function ( char *output_string, int num_char, int *result ).
```

This allows the programmer to include information on the parameter meaning in addition to the parameter type. The above two examples are identical to the compiler.

10.4.4.3 Additions to K&R C

There are several additions to the language. Below is a brief list of the additions to the language:

- #error, #pragma, and #elif to the language.
- The void type.
- The keyword signed.
- The long double type.
- The const and volatile qualifiers.
- The enum data type.

10.4.4.4 Changes to K&R C

The ANSI committee did not completely keep compatibility with K&R C. Minor differences exist. Some of the differences are :

- ANSI C allows the distinction between binary and text files, while K&R C does not.
- ANSI C does not allow macro redefinition while K&R allowed the redefinition but left the behavior as implementation specified.
- ANSI C does not allow recursive macros.
- ANSI C allows <operator>= but not =<operator>.
- ANSI C does not allow int and pointers to be interchangeable.
- ANSI C allows white spaces around #.
- ANSI C changes some library functions specific to UNIX, such as ***kill()*** and ***unlink()***.

Appendix A - Bibliography

[ALI86] Ali, J.K., and M.S. Haniff, "Designing a Real-Time Executive for a Laboratory Microcomputer," *Intelligent Instruments and Computers*, September/ October 1986, p. 232.

[ALON88] Alonso, Rafeal, "Sharing Jobs Among Independently Owned Processors," *Proceedings of IEEE Conference on Distributed Computing Systems,* May 1986, p. 282.

[ATLA89] Atlas, Alan, and Bill Blundon, "Time to Reach for it All," *UNIX Review*, Vol. 7, No. 11, 1989, p. 49.

[BARA86] Baradello, C.S., and G. Carloni, "Generation of Real-Time Executive Systems," *Electrical Communications,* Vol. 60, No. 3/4, 1986, p. 259.

[BERE85] Berets, James C, Ronald A. Mucci, and Richard E. Schantz, "Cronus: A Testbed for Developing Distributed Systems," *Proceedings of IEEE Military Communications Conference*, October 1985, p. 20.4.1.

[BICK88] Bicknell, Paul A., "Software Development and Configuration Management in the Cronus Distributed Operating System," *Proceedings of IEEE Conference on Software Maintenance*, March 1988, p. 143.

[BOTC88] Botcherby, K., Y. Trinquet, and J.P. Elloy, "Event Management and Rendezvous Concept in a Distributed Real Time Operating System," *Proceedings of Distributed Computer Control Systems,* September 1988, p. 43.

[BUEL88] Buell, Robert K., "The Application Of a Distributed Computing Architecture To A Large Telemetry Ground Station," *Proceedings of International Telemetering Conference*, October 1988, p. 571.

[BUTZ89] Butzen, Fred, "Porting to ANSI C," *Unixworld*, Vol. VI, Nos. 5/6, 1989, p. 105 and 109.

[CASE88] Casey, Thomas A., Stephen T. Vinter, D.G. Weber, Varadarajan Rammohan, and David Rosenthal, "A Secure Distributed Operating System," *Proceedings of IEEE Symposium on Security and Privacy*, April 1988, p. 27.

[CHAS88] Chase, Robert P. Jr., David B. Leblang, and Howard Spilke, "Parallel Building: Experience with a CASE System for Workstation Networks," *Proceedings of IEEE Conference on Computer Workstations*, April 1988, p. 2.

[CHEN88] Cheng, Pau-Chen, and Virgil D. Gligor, "A Model For Secure Distributed Computations In A Heterogeneous Environment," *Proceedings of IEEE Aerospace Computer Security Applications Conference*, May 1988, p233.

[CHER86] Cheriton, David R., "Problem-oriented Shared Memory: A Decentralized Approach to Distributed System Design," *Proceedings of IEEE Conference on Distributed Computing Systems,* May 1986, p. 190.

[CICI88] Ciciani, Bruno, Daniel M. Dias, and Philip S. Yu, "Load Sharing in Hybrid Distributed - Centralized Database Systems," *Proceedings of IEEE Distributed Computing Systems Conference,* January 1988, p. 274.

[CLAU87] Clausing, Brian, "Designing a Master Executive for a Distributed Multiprocessor Avionics System," *Proceedings of IEEE National Aerospace and Electronics Conference*, May 1987, p. 174.

[DASG88] Dasgupta, Partha, Richard J. LeBlanc Jr., and William F. Appelbe, "The Clouds Distributed Operating System: Functional Description, Implementation Details and Related Work," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 2.

[DEAN87] Dean, Michael A., Richard M. Sands, and Richard E. Schantz, "Canonical Data Representation in the Cronus Distributed Operating System," *Proceedings of IEEE Conference on Computer Communications*, April 1987, p. 814.

[FERG88] Fergenson, Donald, Yechiam Yemene, and Christos Nikolaou, "Microeconomic Algorithms for Load Balancing in Distributed Computer Systems," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 491.

[FISH89] Fisher, Sharon, "OSI Takes on TCP/IP," *Unixworld*, Vol. VI, No. 2, 1989, p. 74

[FITZ87] Fitzgerald, Ted, "Executive Functions," *Systems International*, September 1987, p. 97.

[GING89] Gingell, Rob, "Shared Libraries," *UNIX Review*, Vol. 7, No. 8, 1989, p. 56.

[GURW86] Gurwitz, Robert F., Michael A. Dean, and Richard E. Schantz, "Programming Support in the Cronus Distributed Operating System," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 486.

[HAGM86] Hagmann, Robert, "Process Server: Sharing Processing Power in a Workstation Environment," *Proceedings of IEEE Conference on Distributed Computing Systems*, May 1986, p. 260.

[HSU86] Hsu, Chi-Tin Huang and Jane W.-S. Lui, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *Proceedings of IEEE Conference on Distributed Computing Systems*, May 1986, p. 216.

[JENS82] Jensen, E. Douglas, "Decentralized Executive Control of Computers," *Proceedings of IEEE Conference on Reliable Distributed System Software*, June 1988, p. 141.

[KNIG88] Knightson, K. G., T. Knowles, and J. Larmouth, "Standards for Open Systems Interconnection," McGraw-Hill, 1988.

[KOVA87] Kovalev, A. S., G.I. Ksenda G. I., and R.B. Shaimardanov, "Design of Distributed Information Processing Systems in Computer Networks," *Avtomatika i Vychislitel' naya Tekhnika*, Vol. 21, No. 2, 1987, p. 71.

[LADD89] Ladd, R. S., "Going from K&R to ANSI C," *Dr. Dobbs Journal*, Vol. 14, No. 8, August 1989, p. 74.

[LEVI89] Levitt, Jason, "In Praise of Korn," *Unixworld*, Vol. VI, No. 6, 1989, p. 67.

[LIN86] Lin, Frank C. H., and Robert M. Keller, "Gradient Model: A Demand-Driven Load Balancing Scheme," *Proceedings of IEEE Conference on Distributed Computing Systems*, May 1986, p. 329.

[LITZ88] Litzkow, Michael J., Miron Livny, and Matt W. Mutka, "Condor - A Hunter of Idle Workstations," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 104.

[LUK88] Luk, W. S., Xiao Wang, and Franky Ling, "On the Communication Cost of Distributed Database Processing," *Proceedings of IEEE Conference on Distributed Computing Systems*, May 1986, p. 528.

[MANT88] Mantelman, Lee, "Upper Layers: From Bizarre to Bazaar," *Data Communications*, Vol. 17, No. 1, January 1988, p. 110.

[MCCA89] McCarron, Shane P., "Standards Report: ANSI C," *UNIX Review*, Vol. 7, No. 7, 1989, p. 12.

[MCCA89] McCarron, Shane P., "Standards Report: Standards and UNIX Security," *UNIX Review*, Vol. 7, No. 11, 1989, p. 21.

[MEHT89] Mehta, Sunil, "Standards Report: User Interface and the IEEE P1201 Committee," *UNIX Review*, Vol. 8, No. 1, 1989, p. 14.

[ORIE89] O'Reilly, Tim, "The Toolkits (and Politics) of X Windows," *Unixworld*, Vol. VI, No. 2, 1989, p. 66

[PITT88] Pitts, David V., "Recovery in the Clouds Kernel," *Proceedings of IEEE Symposium on Reliable Distributed Systems*, October 1988, p. 167.

[PITT88] Pitts, David V., and Partha Dasgupta, "Object Memory and Storage Management in the Clouds Kernel," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 10.

[PLAU89] Plauger, P. J., and Jim Brodie, "Standard C," Microsoft Press, 1989.

[POPE81] Popek, G., B. Walker, J. Chow, and D. Edwards, "LOCUS: A Network Transparent, High Reliability Distributed System," *Proceedings of IEEE Symposium on Office Systems Principles*, December 1981, p. 169.

[PRAT85] Pratt, Keith D., and Roy L. Sherrill, "Experiences with the Development of a Real-Time Multiprocessor Executive in Ada," *Proceedings of IEEE National Aerospace and Electronics Conferences*, May 1985, p. 672.

[SCHA86] Schantz, Richard E., Robert H. Thomas, and Girome Bono, "The Architecture of the Cronus Distributed Operating System," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 250.

[SERL89] Serlin, Omri, "The Serlin Report: AT&T Strikes Back," *Unixworld*, Vol. VI, No. 3, 1989, p. 33

[SIMP89] Simpson, David, "V.4 vs. OSF/1: What's the Difference?," *Systems Integration*, May 1989, Vol. XXII, No. 5, p. 53.

[SIMP89] Simpson, David, "Will the Real-Time UNIX Please Stand Up?," *Systems Integration*, Vol. XXII, No. 12, December 1989, p. 46.

[STAL87] Stallings, William, "Handbook of Computer-Communications Standards," Howard W. Sams & Company, 1987.

[STAN86] Stankovic, J. A., and D. Towsley, "Dynamic Reallocation in a Highly Integrated Real-Time Distributed System," *Proceedings of IEEE Conference on Distributed Computing Systems,* May 1986, p. 374.

[STUM88] Strumm, Michael, "The Design and Implementation of a Decentralized Scheduling Facility for a Workstation Cluster," *Proceedings of IEEE Conference on Computer Workstations,* March 1988, p. 12.

[SUSS89] Suss, Warren H., "The Government Open Systems Interconnection Profile (GOSIP)," *Datapro Management of Data Communications,* McGraw-Hill, 1989, p. CS93-107-401.

[THEI88] Theimer, Marvin M., and Keith A. Lantz, "Finding Idle Machines in a Workstation-based Distributed System," *Proceedings of IEEE Distributed Computing Systems Conference,* January 1988, p. 112.

[THOM84] Thompson, William B. Jr., "An Integrated Approach to Telemetry Ground Stations Using Distributed Processing," *Proceedings of International Telemetering Conference*, October 1984, p. 371.

[THOM89] Thomsen, Kristine Stougaard and Jorgen Lindskov Knudsen, *Tutorial: Distributed Software Engineering,* 1989, p. 122.

[VINT86] Vinter, Stephen, Krithi Ramamritham, and David Stemple, "Recoverable Actions in Gutenberg," *Proceedings of IEEE Conference on Distributed Computing Systems,* May 1986, p. 242.

[VANH89] van Halm, Rochelle, "Real-time in the Real World," *Unixworld*, Vol. VI, No. 9, 1989, p. 58.

[WALP88] Walpole, J., G.S. Blair, J. Malik, and J.R. Nicol, "Maintaining Consistency in Distributed Software Engineering Environments," *Proceedings of IEEE Distributed Computing Systems Conference*, January 1988, p. 418.

[WANG87] Wang, Jingwen, "The Design and Study of the Kernel Executive for DRIPS, a Distributed Real-Time Information Processing System," *Proceedings of IEEE Second International Conference on Computers and Applications,* 1987, p. 208.

[WARI88] Waring, W. M., and S.S. Sinor, "A User's Perspective: Distribute Architecture Software Development," *Proceedings of IEEE National Aerospace and Electronics Conference*, May 1988, p. 608.

This section includes the interim report given to NASA-JSC after the executive technology survey and requirements definition phases were complete. The presentation includes information on the Concept Executive and all executive systems, user interfaces, and standards surveyed.

SwRI

NASA

NASA Grant NAG 9-340
Interim Report

Presented to:
NASA-JSC

Presented by:
Mark D. Collier
William A. Hoover
Nancy L. Martin

Southwest Research Institute
6220 Culebra Road
San Antonio, Texas 78228

Research into
Software Executives

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Introduction

Many computing environments are transitioning to use of graphics workstations as application processors in a distributed system.

Although distributed processing has been used for some time in real-time systems, workstations have been avoided for reasons of performance and limitations of UNIX.

It is now feasible to use graphic workstations in a real-time distributed computing environment. This however requires development of a "workstation executive" which provides:

- Standardization.
- System integration and application support.
- Environment control.

This research grant focuses on identification of software technology which allows these goals to be achieved.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Research Background

NASA Grant NAG 9-340, which is entitled "Research in Software Executives for Space Operations Support", is a continuation of efforts to investigate, research, and apply new technologies applicable to workstation executives.

Prior efforts in this area include the Hardware Independent Software Development Environment (HISDE). HISDE introduced:

- Exclusive use of software standards including the C language, System V UNIX, X Windows, ISO communications, and GKS.
- Flexible access to the UNIX file system and the UNIX shell.
- A more flexible, X Windows-based user interface.
- An alternative configuration management scheme.

This grant is proceeding in conjunction with NAG 9-388, which is entitled "Continuation of Research in Software for Space Operations Support".

Primary Research Directions

Phase 1 - Survey government, commercial, and research sites for systems and technologies relevant to workstation executives. This includes evaluation of:

- Operational and proposed telemetry processing/executive systems.
- Other systems and technologies relevant to workstation executives.
- Current and proposed software standards.

Phase 2 - Using the information gathered during the technology survey, perform the following:

- Define an "executive".
- Define the scope of a workstation executive within the target control environment.
- Define a set of functional requirements which specifies a "Concept Executive.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Primary Research Directions (Continued)

Phase 3 - Develop prototypes of the concepts identified in the first two phases. At the current this time, the following efforts have been identified:

- Intelligent process distribution and fault tolerance.
- Support of multiple workstation configurations (configuration independence).
- Data acquisition and distribution in such configurations.
- A real-time graphic text display mechanism.

Research Status:

- Phases 1 and 2 are complete.
- Prototypes for phase 3 have been identified and research is already underway.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

The first task of Phase 1 is to define an "executive". The classic definition of an executive is:

- A low-level software subsystem which provides programmers a means of real-time data access and system control. Examples are Vertex and VxWorks real-time executives.

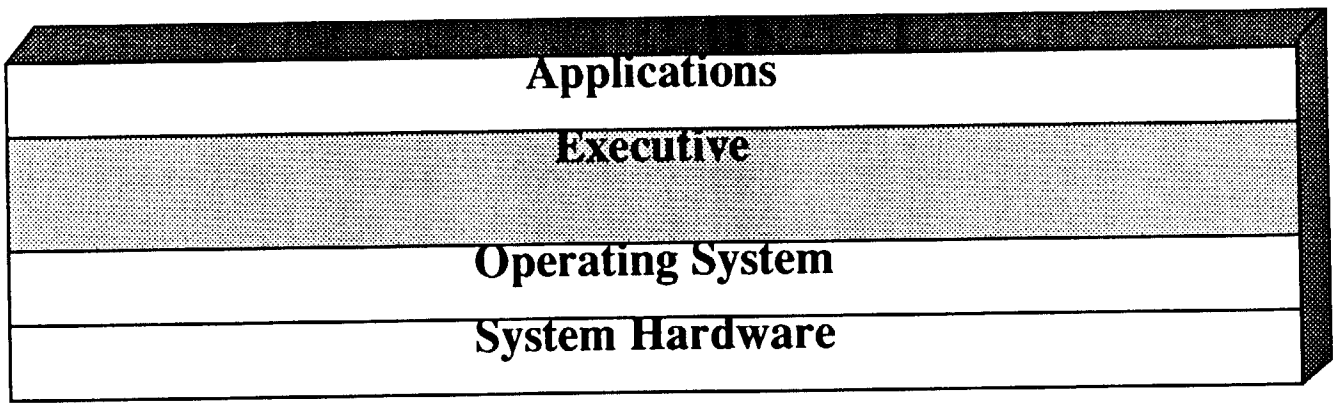
Characteristics of an executive:

- Provides services to application programmer (makes the job of programming simpler and more application-specific).
- Fast, efficient, and robust.
- Provides a limited, application-specific set of services.
- The terms "executive" and "real-time" are effectively inseparable.
- Is not a complete operating system.

New Definition:

- A software subsystem which adds upon the inherent hardware and software functionality to meet the requirements of the environment and its application programmers.

Location of an executive in system hierarchy:



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

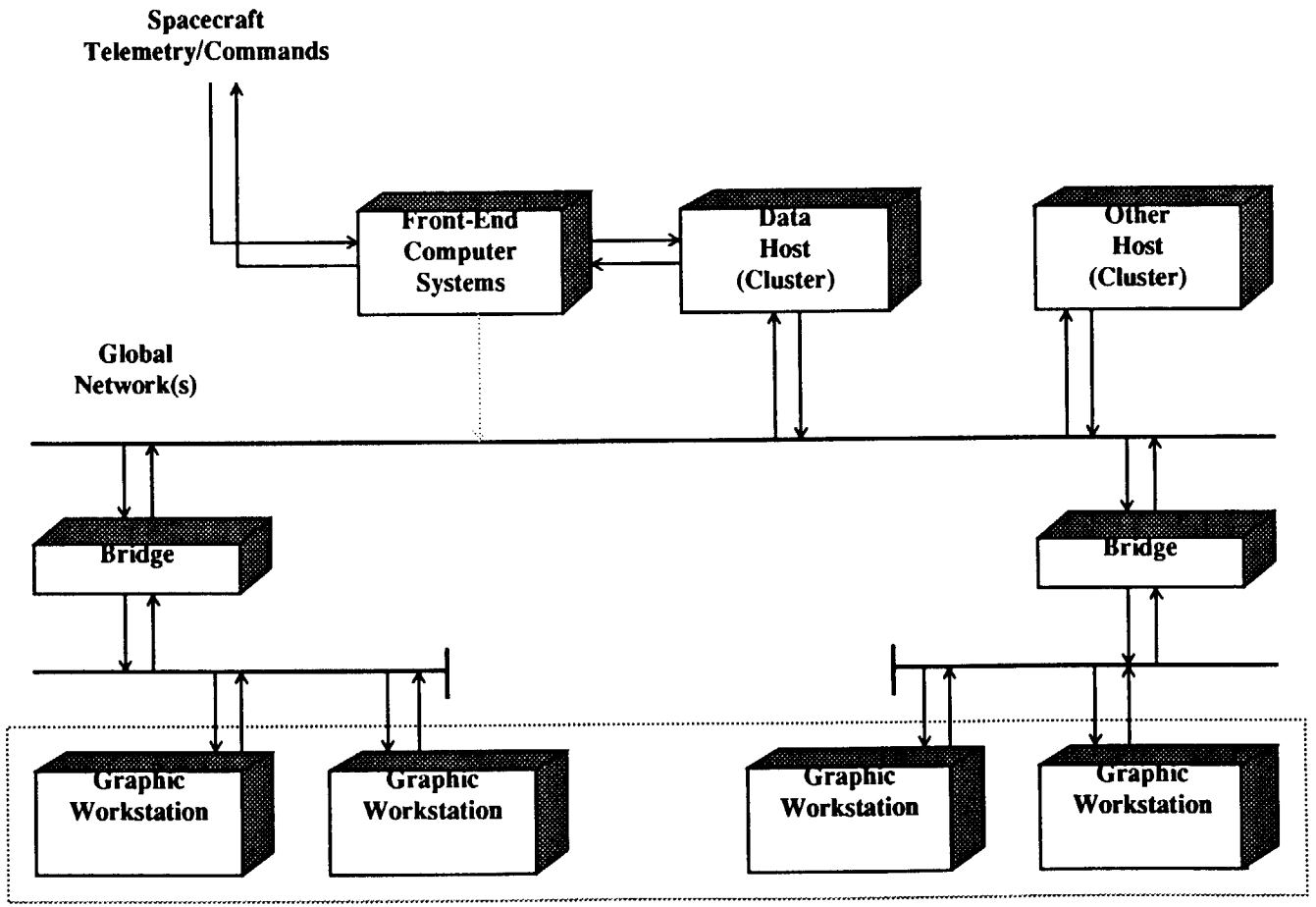
Target Environment Characteristics

In order to define the scope of an executive, it is necessary to define the target environment. The target processing environment is characterized by:

- Telemetry processing.
- Use of workstations as application processors.
- Use of one or more high-speed networks to provide workstation connectivity.
- Real-time throughput of data is required.
- Workstations will use high-performance graphics for user interaction and display.
- Host support will be provided for logically centralized functions.
- Configuration management and security are required.
- Emphasis placed on use of COTS and standard software.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Target Environment Hardware Diagram



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive Introduction

The Concept Executive is a high-level set of functional requirements which defines a generic system within the target environment. The Concept Executive is intended to be expandable to support multiple control center environments.

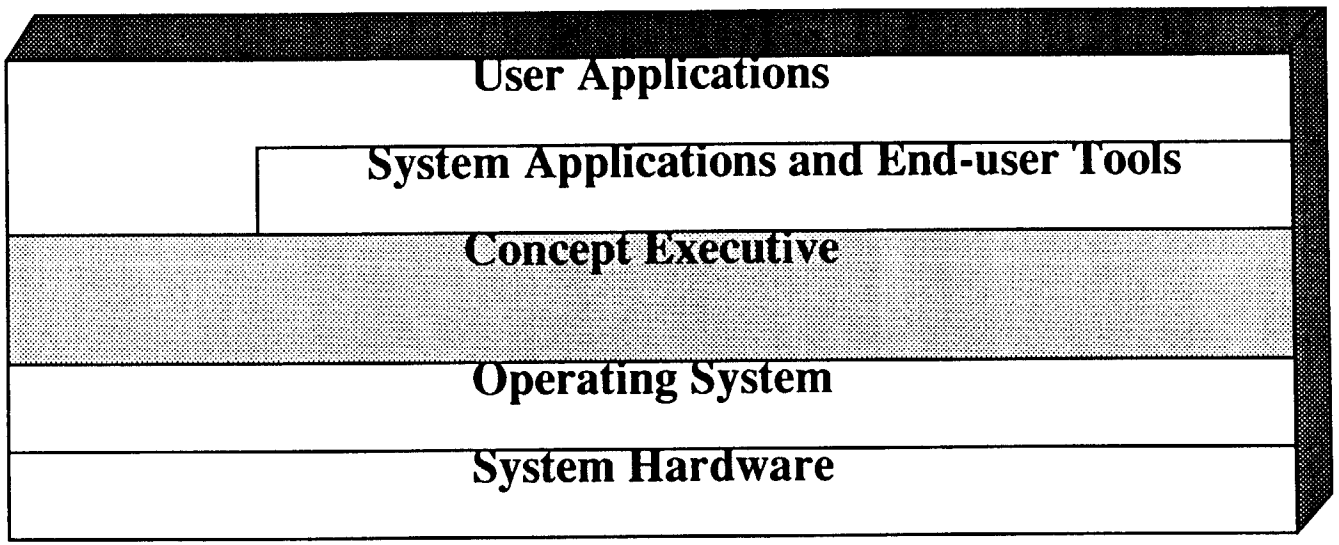
The primary goal is to provide the baseline system which supports multiple environments and will have a life cycle extending far into the future.

The Concept Executive provides a COTS and custom-based software environment which addresses the requirements of application programmers.

Application programmers are expected to address the specific requirements of users in different environments.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive Location in System

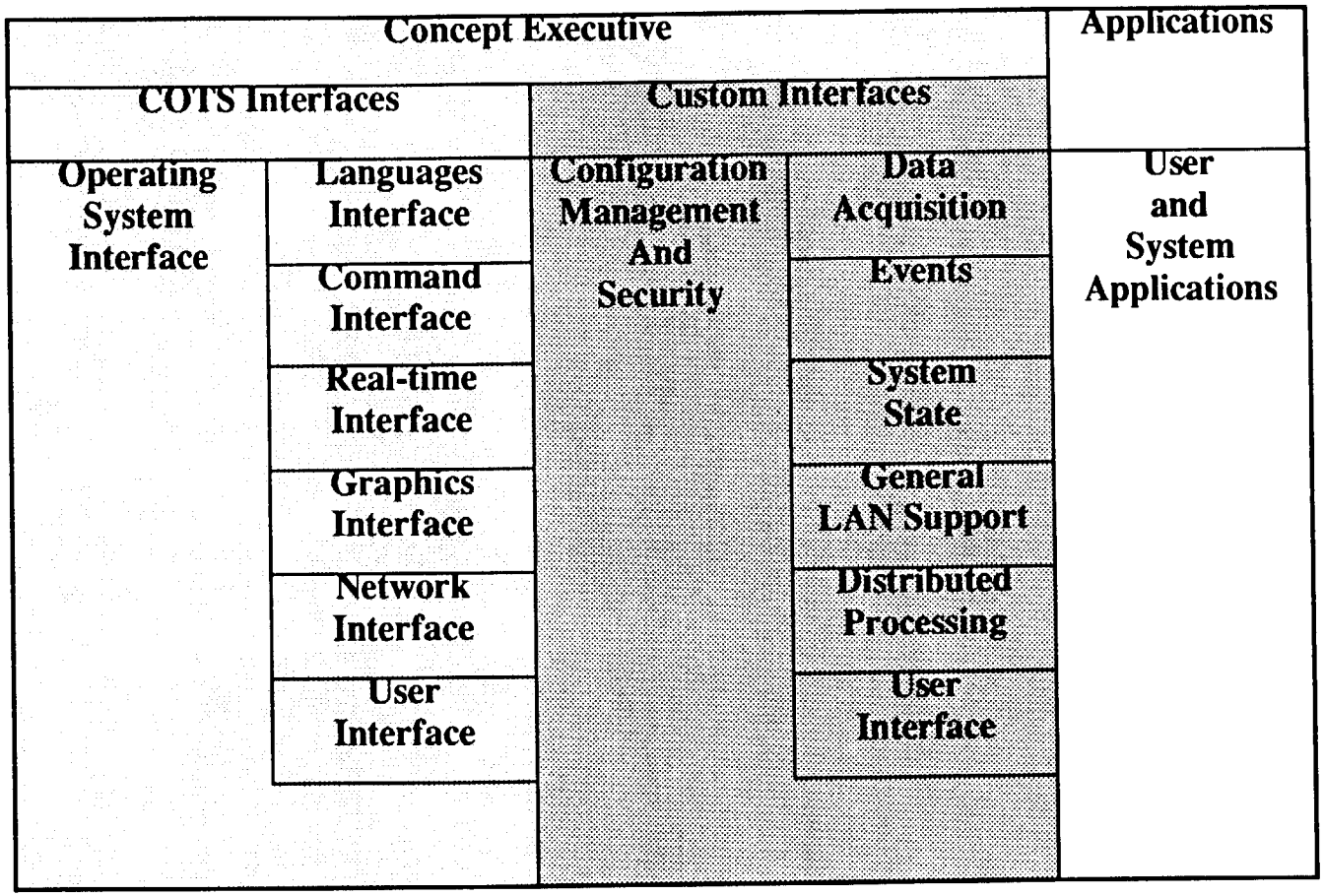


NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

SwRI

Concept Executive
COTS and Custom Contents

NASA



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive
Standardization/Environment Control

Standardization

- Use of standard software.
- Isolation of hardware/vendor dependent functions.
- Configuration independence.
- Migration to future standards.

Environment control and support:

- Configuration management.
- Security.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

System Integration and Application Support

- Processing of flight data.
- Networking support.
- Graphics user interface support.
- Programming languages.
- Command line interpreter.
- Distributed processing.
- Events.
- System state.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive Functionality Assumptions

Functions assumed to be globally performed by the system:

- Execution of simulations.
- Network(s) mode selection.

Functions assumed to be performed by data distribution systems:

- Real-time data generation.
- Real-time data retention, archiving.
- Generation of general data.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive
Functionality Assumptions (Continued)

Functions assumed to be performed by hosts or dedicated workstations:

- Spacecraft command verification.
- Network management.
- Health and status.
- Configuration management (CM) workstation.
- Certified application archive.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Concept Executive
Functionality Assumptions (Continued)

Functions present on the workstation, but not performed by the executive:

- Simplification functions.
- Data-driven displays.
- Custom command languages.
- Host function interfaces.
- A complete window-based user interface.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Overall System Requirements

- Workstation based distributed processing.
- Generic environment support.
- Identical operational environment.
- Workstation hardware expandability and interoperability.
- Real-time response.
- Use of Commercial-Off-The-Shelf software.
- Use of standard software.
- Migration to POSIX.
- Isolation of all non-standard functions.
- Executive portability.
- Application portability.
- Modifications to COTS software.
- Software support.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Overall System Requirements (Continued)

- Configuration independence.
- Global and local network independence.
- Data throughput increases.
- Modes of operation.
- Support of multiple modes.
- Concurrent flight (operation) support.
- Continuous duration flight support.
- Programmatic and command level interfaces.
- Support of ASCII terminals.
- Resource utilization.
- Application termination.
- Error handling and reporting.
- On-line help.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

COTS Software Subsystems Introduction

The COTS software subsystems making up the Concept Executive include:

- Operating system interface
- Command line interface.
- Graphic user interface.
- Graphic plotting and modeling interface.
- Programming language interface.
- Networking.
- Real-time extensions.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

COTS Software Subsystems
Operating System

It is assumed that the operating system will be a UNIX variant. At this time, two operating systems will vie for share of UNIX market:

- System V Release 4 (SVR4) from ATT/UNIX International.
- OSF/1 from the Open Software Foundation.

SVR4 is specified as the operating system for the Concept Executive for the following reasons:

- Consists of a merge of System V Release 3, BSD 4.3, and Xenix.
- Will be available in early 1990.
- Compatible with SVR3, BSD 4.3, and Xenix programs.
- Satisfies upgrade path to POSIX.
- Offers some real-time capability.
- Includes ANSI C.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

COTS Software Subsystems
Command Line Interface Requirements

The Concept Executive will provide the native command line interpreter (the shell) as the command interface:

- Represents a familiar interface for programmers.
- COTS product.
- Shell is adequate and appropriate for application programmers.
- Use controllable via configuration management.
- Applications can still provide environment and user-specific interfaces.

A custom interface is not specified because:

- A new interface would be unfamiliar to programmers.
- A new interface would be expensive and time-consuming to develop.
- A new interface would add overhead and slow response time.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Recommended shell standard is the Korn shell (ksh): The Korn shell is:

- More efficient than the C shell and almost as efficient as the Bourne shell.
- Available as part of SVR4.
- Fully upgrade compatible with the Bourne shell.
- Includes many interactive features including command history, aliases, and command editing.
- Compatible with the direction of POSIX 1003.2.

COTS Software Subsystems
Graphics User Interface Requirements

The graphics user interface provided by the Concept Executive shall be the foundation for all interaction. The interface shall be:

- Completely standards based.
- Based on the client-server model.
- Network transparent.

The complete graphic user interface shall consist of the following libraries and applications:

- A low-level, primary interface.
- A user interface toolkit.
- A high-level user interface language (UIL).
- An accepted look and feel specification.
- A window manager.
- A set of interactive applications.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Low level interface:

- Xlib library.

User interface toolkit:

- Xt library.
- Motif widgets.

User interface language:

- Motif UIL.

Look and feel:

- Motif look and feel specification.

Window manager:

- Motif window manager (mwm).

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Interactive clients:

- No standard available at this time. Assume availability of xterm, xset, xsetroot, host, xwd, xwud, xpr, xlsfonts, xfd, xdpyinfo, xkill, xwininfo, xlswins, xmodmap, xrdb, xrefresh, and bitmap clients

COTS Software Subsystems
Graphic Plotting and Modeling

Graphic plotting and modeling software is required for more straight-forward drawing and graphic display. The primary requirements are:

- Must be well-integrated with the graphics user interface.
- All graphics must be rendered through the same mechanism as used for the graphics user interface.

Recommended standards:

- Graphics Kernel System (GKS).
- Programmers Hierarchal Interactive Graphics System (PHIGS and PHIGS+).

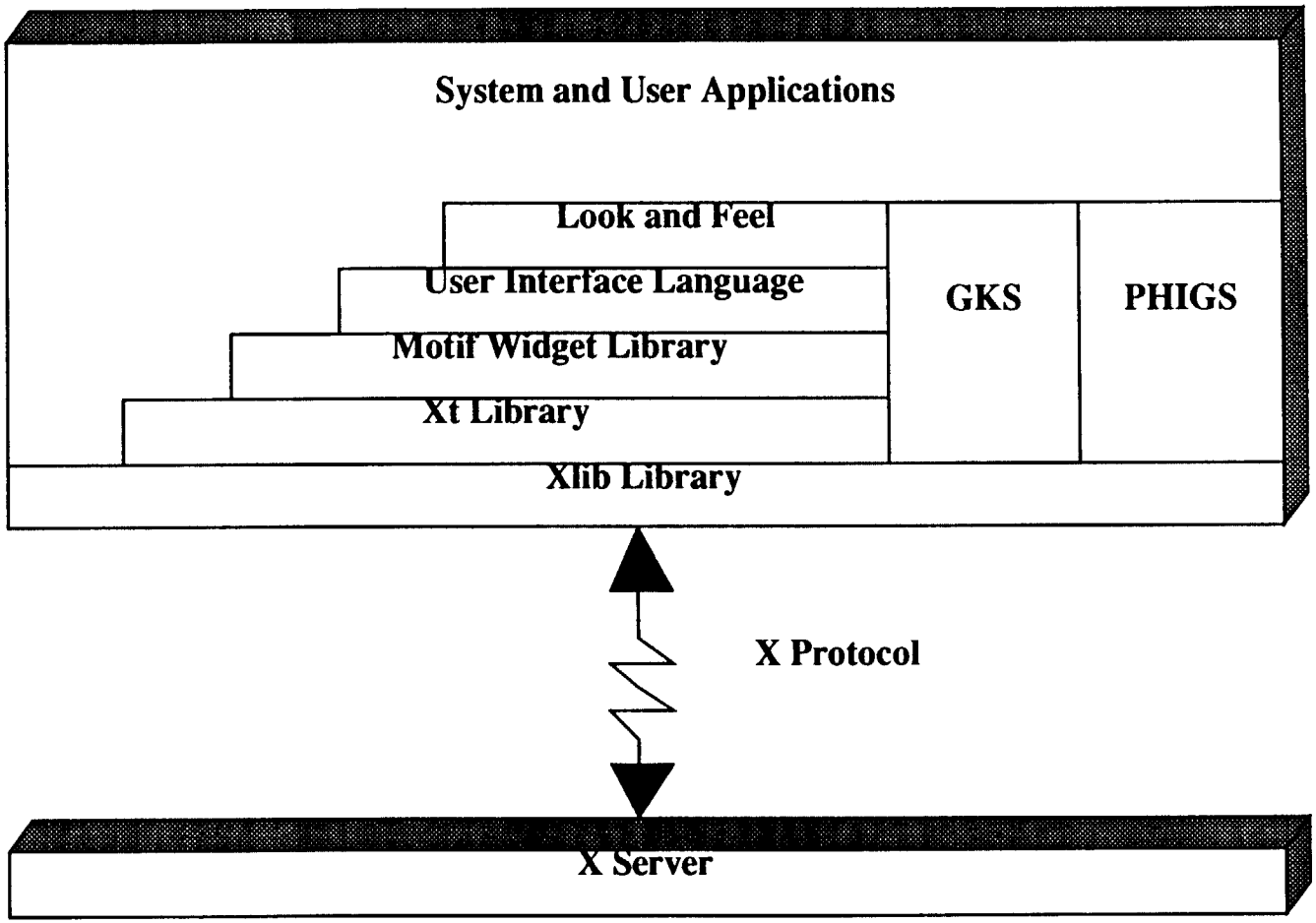
Recommend use of system most suitable for current environment.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

SwRI

COTS Software Subsystems
Graphics Software Overview

NASA



Research into
Software Executives

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

COTS Software Subsystems
Programming Language Reqs./Standard

The language provided by the Concept Executive must provide:

- Industry-wide acceptance.
- Development of portable applications.
- Support for system and application development.
- Functionality in UNIX.
- Support interfaces to COTS and Concept Executive functions.

Recommended Standard is ANSI C. The advantages are:

- Compatibility with K&R C.
- Supports above requirements.
- POSIX compatible.

The Concept Executive allows use of other languages, but does not guarantee availability of all interface bindings.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

COTS Software Subsystems
Networking Requirements/Standard

The networking support provided by the Concept Executive must provide:

- Connection based communications.
- Support for distributed processing.
- Electronic mail.
- Directory services.
- Remote file sharing.
- Remote login.
- Adherence to OSI seven layer model (only level 4 or above communications allowed).

Recommended standard is POSIX 1003.8:

- Will be compatible with ISO and GOSIP.
- NFS satisfies current requirements of remote file sharing.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

The real-time support provided by the Concept Executive must provide:

- Priority scheduling.
- Asynchronous event notification.
- Message passing.
- Process memory locking.
- Shared memory.
- Binary semaphores.
- Threads.
- Timers.
- Real-time synchronous and asynchronous I/O.
- Real-time files.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Recommended standard is POSIX 1003.4.

- Not yet completed.
- No other standard exists or is in development.
- Compliance with 1003.4 does not insure suitability to task.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom software subsystems include:

- Configuration management.
- Security.
- User interface.
- Data acquisition.
- General-purpose communications.
- System state.
- Events.
- Distributed processing.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

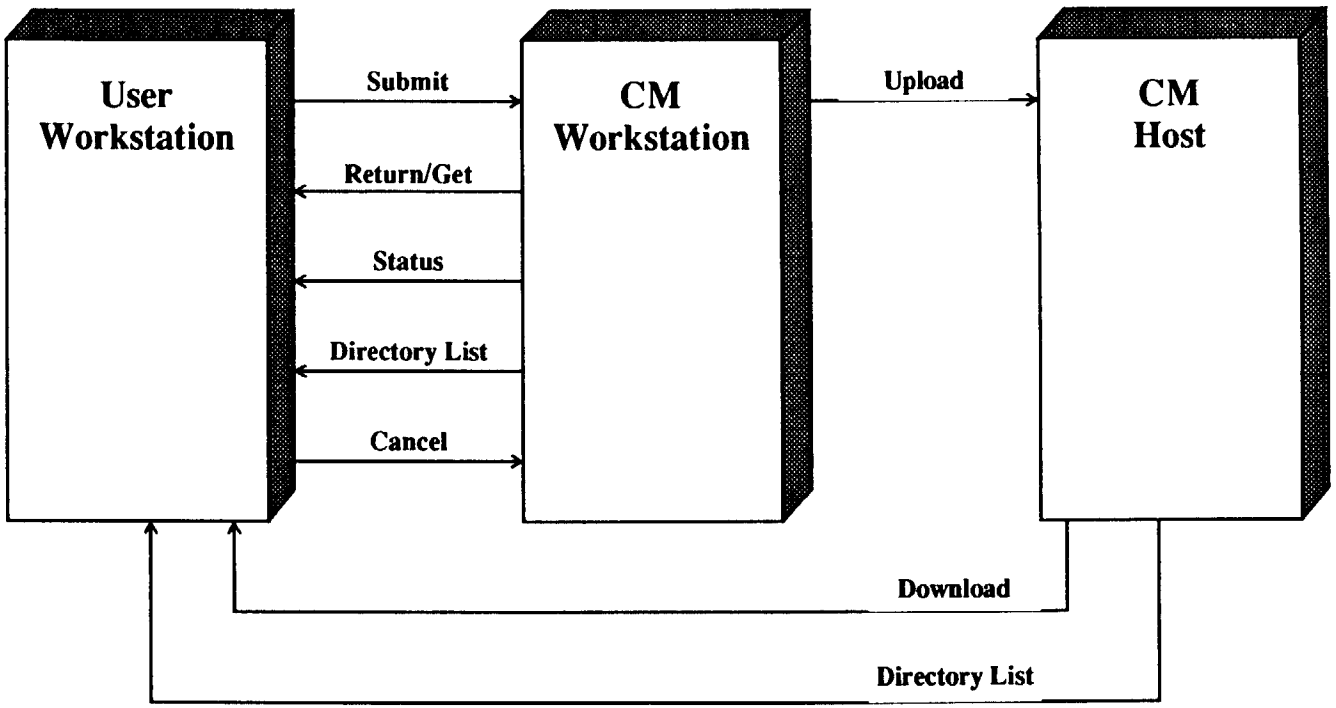
Custom Software Subsystems Configuration Management

The Configuration Management subsystem shall provide the following functions:

- Access to operations-certified libraries.
- A certified environment for operations mode.
- A method for downloading a list of certified files.
- Protection for the global networks during development and simulation modes.
- Restricted command access during operations mode.
- Protection of the Concept Executive during execution.
- Controlled access to the root permission.
- Method for integrating new workstations and software revisions into the network.
- Commands to handle the submission, retrieval, upload, download, etc. of all certified software.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

A CM workstation and a CM host will be necessary to develop certified software which will in turn be downloaded to a workstation for operation.



Custom Software Subsystems
Security

The Security subsystem shall define a secure distributed system which provides the following:

- User account security.
- User account administration.
- Limited super user capability.
- Process execution security.
- User privilege restrictions.
- Command issuance security.
- Proprietary data protection.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
User Interface

The User Interface Subsystem shall present an environment which is consistent across all operating modes. This subsystem will provide:

- A login application to control access to the system.
- A graphic interactive environment with a consistent look and feel.
- A real-time mechanism for the display of dynamic text.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

In support of configuration management, a new Login client will replace the existing UNIX process. This client will:

- Present an easy-to-use interface.
- Allow entry of user identification information.
- Allow selection of environment-specific information necessary for the initialization of the session.
- Allow specification of the access mode.
- Allow specification of special environment initialization files.

The User Environment shall provide:

- A simple interactive environment for all levels of users.
- Interaction with the UNIX shell for the execution of commands.
- A consistent graphic user interface behavior through the selection of the Motif widget set.
- A consistent look and feel through standardized interaction constructs, such as colors, mouse inputs, and the use of menus, etc...
- One window manager to maintain a consistent look and feel throughout the system.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
User Interface (Real-time Display)

A real-time display mechanism shall allow the real-time display of large quantities of data values by utilizing the operating system's real-time features and the lowest level of the X Windows interface. This mechanism shall:

- Allow specification of individual, or a buffer of, values to be used for update.
- Only update values which have changed.
- Allow specification of the display action taken for unexposed areas of the window.
- Allow the specification of colors and fonts for limit conditions.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

The Data Acquisition subsystem of the Concept Executive shall provide:

- A uniform set of interfaces to all network data.
- No loss of data due to workstation response time.
- A table driven map of the data.
- "N" most recent sets of real-time data.
- Support more than one stream of real-time data.
- Real-time data in display workstation memory.

Custom Software Subsystems
General-purpose Communications

The General-purpose communications subsystem shall provide the following:

- Consistent set of interfaces to communicate over the system networks.
- Support workstation to workstation, to and from a host computer, and displays between workstations.
- Support for broadcast, multicast, and point-to-point transfers.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
System State Subsystem

The System State subsystem shall provide access to and control of the environment-specific variables which affect the operation of the executive. This includes the following:

- Active flights and current flights.
- Host or locally generated GMT.
- Current user and group (flight position).
- Access mode of workstation.
- Access mode of network and hosts.
- Default host for communications
- Security status.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Event Subsystem

The Event Subsystem is responsible for queueing of all events generated by the workstation and received from the network.

An event is defined as a string of ASCII text which has intrinsic meaning to a user. The Event Subsystem is the central point from which this type of data is processed.

Functions will be provided to:

- Retrieve last event.
- Retrieve last event by type.
- Retrieve last event by source.
- Enable or disable logging to a file
- Specify event log file or device.
- Clear event queue.
- Generate local or remote (other workstation or host) event.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Distributed Processing Subsystem

The Distributed Processing Subsystem provides an integrated set of functions which support distribution and control of processes. The functions to be provided include:

- An interface which provides status information.
- Process distribution and load balancing.
- Automatic fault tolerance and process migration for critical processes.
- Transparent (to application programmer and user) support of different workstation configurations (configuration independence).

The Distributed Processing Subsystem will:

- Adhere to the rules of configuration management.
- Allow definition of static and dynamic processor subsets.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Health and Status information will be required to provide the statistic base for distributed processing. The information to be retrieved includes:

- CPU percentage.
- Memory utilization.
- Disk operations on all controllers and disk drives.
- System interrupts, system calls, and context switches.
- Process specific information.
- Graphics activity.

Process distribution and load balancing will allow processes to be automatically executed on processors with the least load. The load will be based on:

- CPU usage.
- Memory utilization.
- Disk operations.
- Graphics operations.

Interfaces will be provided to:

- Execute a process on workstation with the lowest load.
- Allow a workstation to enable or disable its participation in load balancing.
- Select the load balancing algorithm and the weight assigned to each load statistic.
- Report current load to allow manual execution of processes.

Application fault tolerance will allow designation of critical applications (processes) and the action to take place if they fail.

If a designated application fails, one of the following actions will take place:

- Re-start the application of local (or remote if necessary) workstation.
- Re-start the application on the workstation with the lowest load.

Configuration independence refers to the ability to support multiple configurations of workstations.

The advantages of configuration independence are:

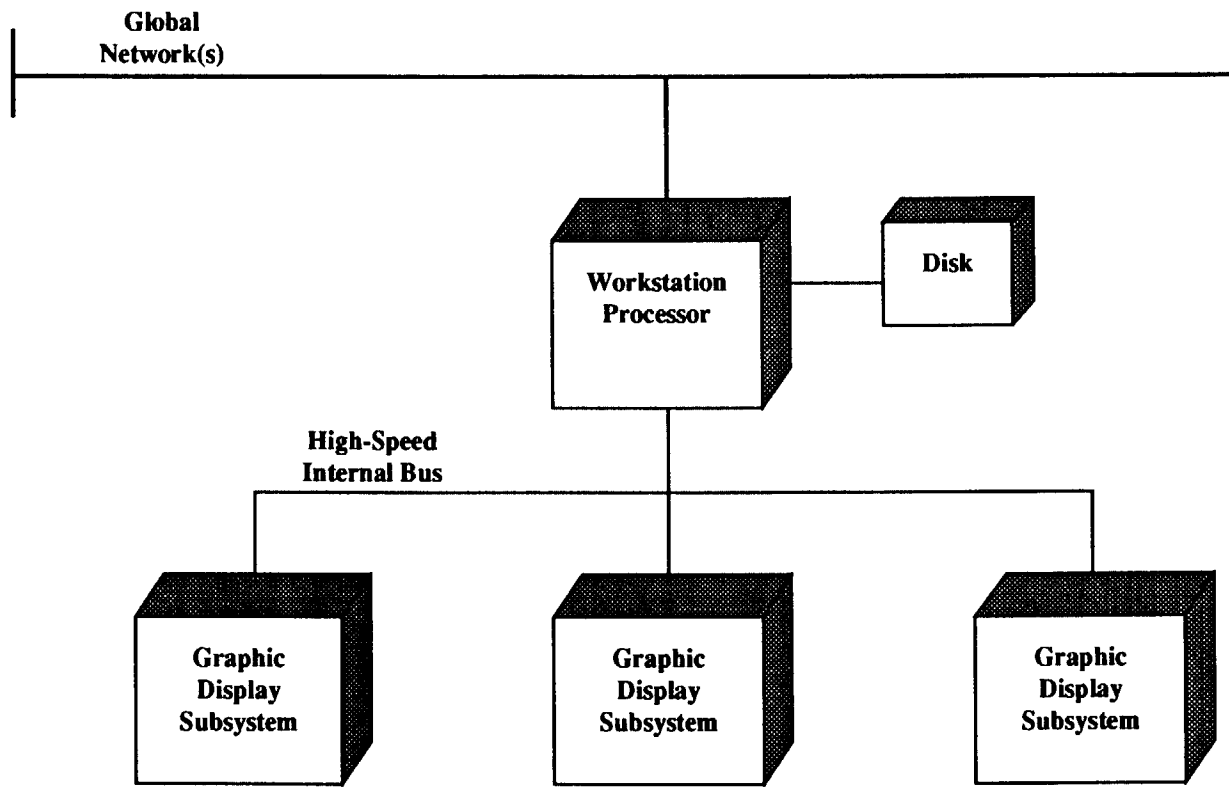
- True hardware independence.
- Resource sharing.

The disadvantages are:

- A more complicated executive.
- Potentially reduced performance.

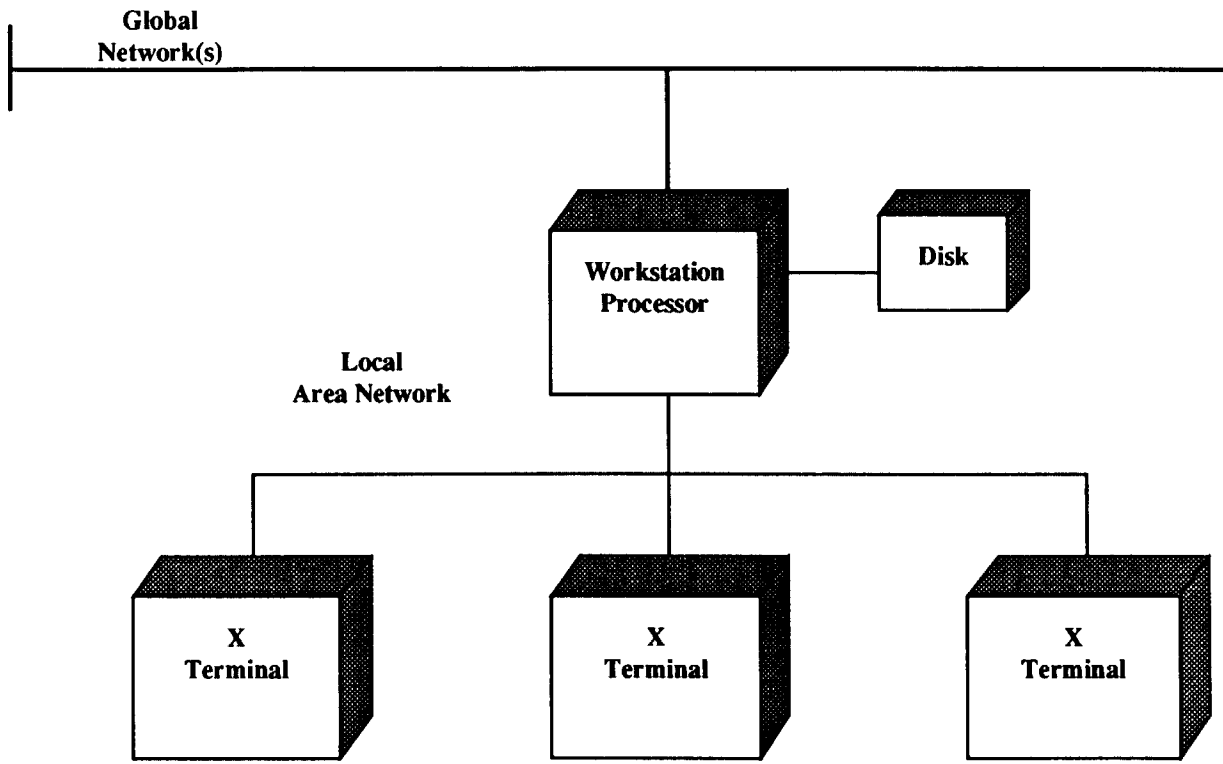
NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Distributed Processing Subsystem (Cont.)



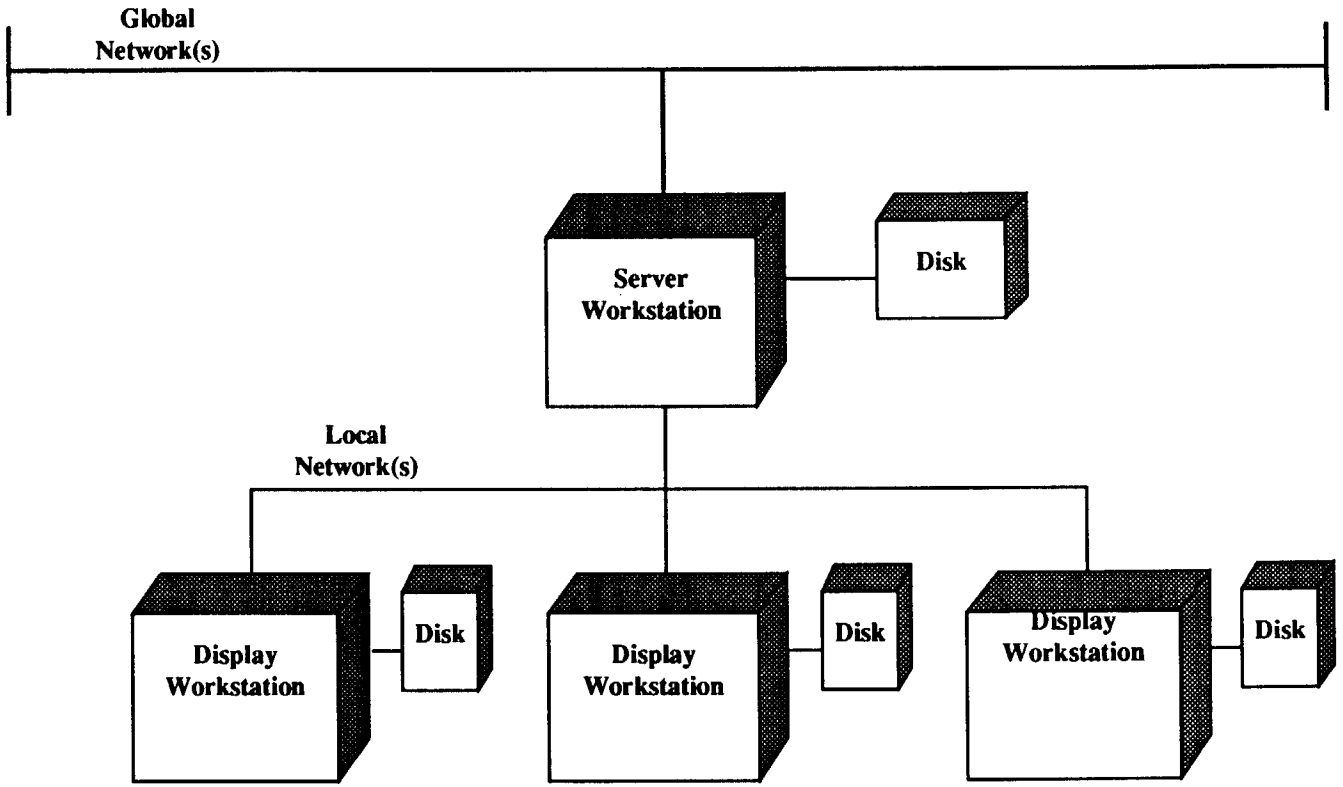
NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Distributed Processing Subsystem (Cont.)



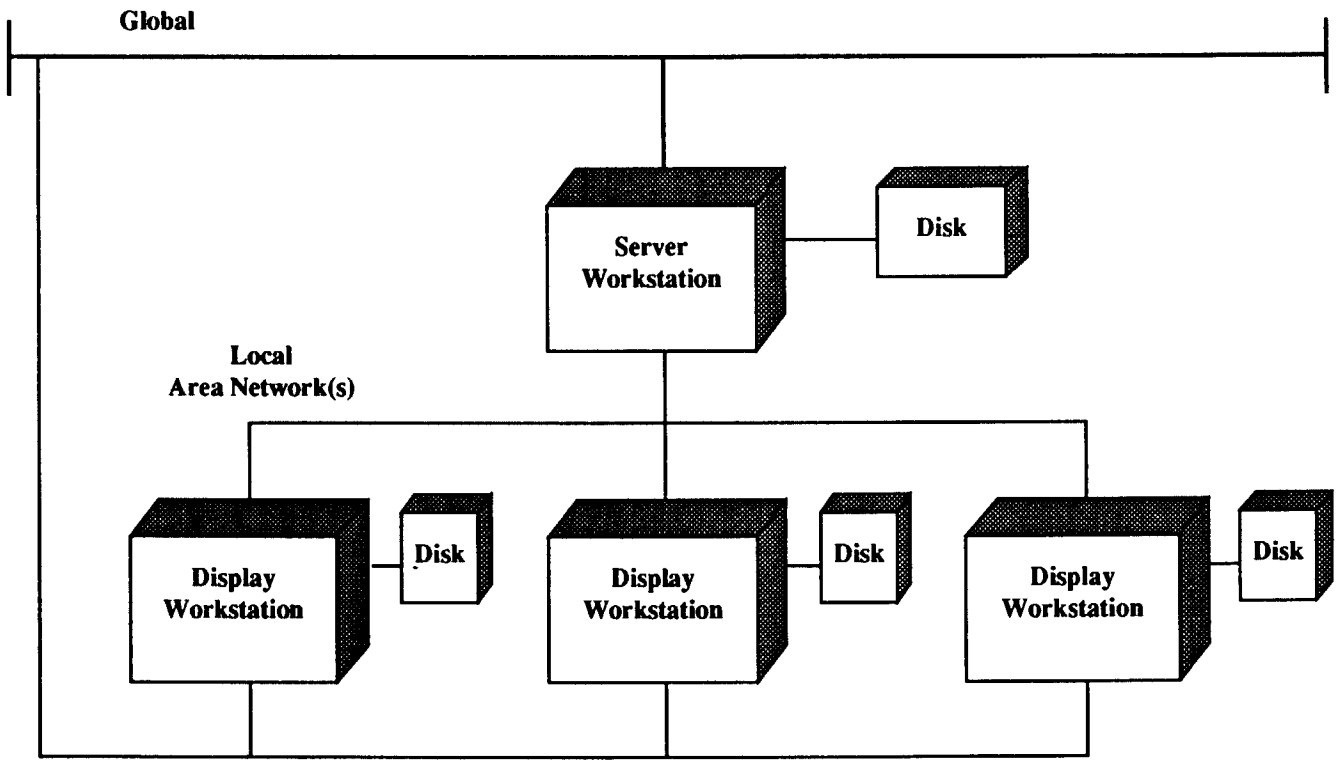
NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Distributed Processing Subsystem (Cont.)



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Custom Software Subsystems
Distributed Processing Subsystem (Cont.)



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

TOAST provides:

- The capabilities required to support a mission in preflight, real time, and postflight modes.
- An integrated set of verified applications which are designed to interface with the LAN data flow.

Its features include:

- Control of user sign-on and database selection.
- A menu tree for selection of the desired applications and displays.
- Application dispatching, resource loading control, and result storage.
- Logging of user input commands sufficient to reconstruct configuration and computation histories.
- A checkpoint capability.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
PPS

The Huntsville Operations Support Center (HOSC) Peripheral Processor System (PPS) is one part of the Payload Operations Control Center (POCC) and provides:

- Digital data recording, distribution, processing, and display.
- Command generation.

Characteristics of the PPS include:

- Highly distributed system.
- Maintains configuration control.
- User has high degree of flexibility over display.
- Tightly coupled to proprietary systems.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
IRAF

The Image Reduction and Analysis Facility (IRAF) is a system maintained by the National Optical Astronomy Observatories (NOAO) and is in use at different observatories. This system is also used to support the Hubble Space Telescope.

IRAF runs on VMS and UNIX systems and provides a complete application development environment. It includes application software tailored towards image processing.

The IRAF system consists of the following:

- A command line interpreter.
- Numerous applications.
- The subset preprocessor language (SPP).
- The virtual operating system.
- The host system interface.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
SFOC (Continued)

The SFOC baseline includes the following 3 core subsystems:

- Workstation Support Environment Subsystem.
- Data Transport Subsystem.
- Common Data Access Subsystem.

The SFOC baseline also includes the following 7 application subsystems:

- SFOC Monitor and Control Subsystem.
- Central Database Subsystem
- Ground Communications Facility Subsystem.
- Telemetry Input Subsystem.
- Data Monitor and Display Subsystem.
- Digital Television Subsystem.
- Test Workstation.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
SFOC

The Space Flight Operations Center (SFOC) is a system in use and under development at the Jet Propulsion Laboratory. It will replace the existing Mission Control and Computing Center (MCCC).

The SFOC system offers the following:

- Use of common hardware and software to support multiple missions.
- Extensive use of standard software including UNIX, C, TCP/IP, and X Windows.
- An improved user interface.
- Use of workstations for many computing nodes.

The SFOC system is currently used to support the Magellan spacecraft. The SFOC system will be used in the future to support Galileo, Mars Observer, Voyager, and Ulysses spacecrafts.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

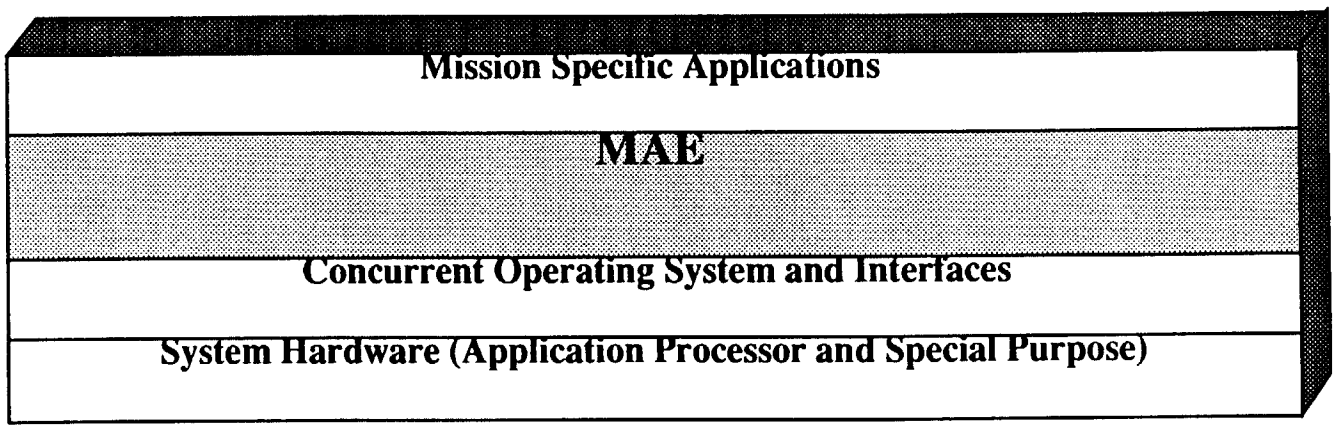
MAE consists of the following software subsystems:

- External interfaces.
- Display.
- System Test and Operations Language (STOL).
- Network Control Center (NCC).
- History.
- Database.
- Library.

The Multi-Satellite Operations Control Center (MSOCC) Application Environment (MAE) is the system currently in use at Goddard Space Flight Center.

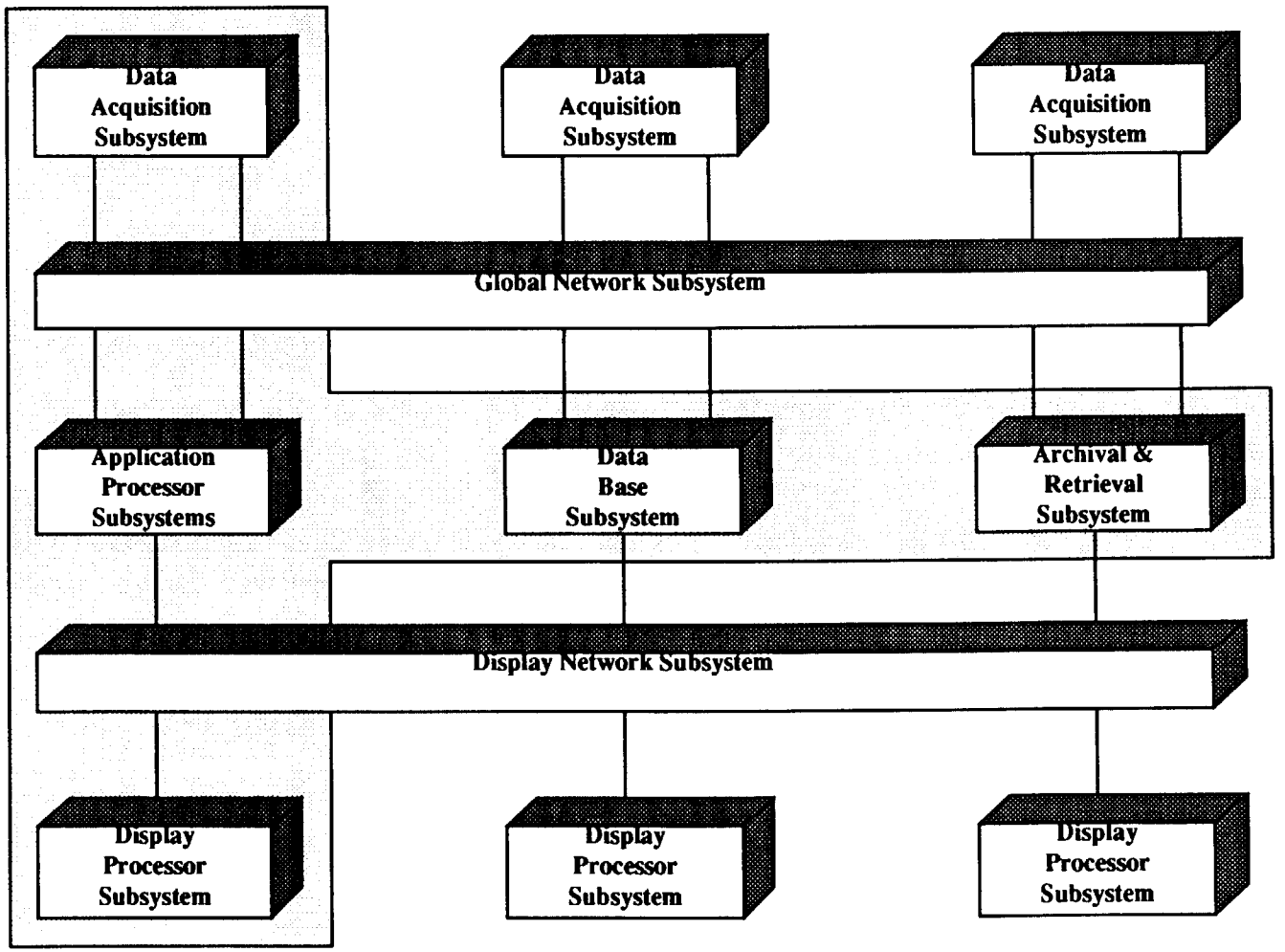
MAE is an executive which provides a generic, primitive set of mission support functions. MAE is used and expanded for each mission to support mission specific requirements.

The location of MAE within the system hierarchy is:



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
GCS (Continued)



NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Survey - Executive Systems
GCS

The Generic Checkout System is a planned system for use at Kennedy Space Center. The GCS system is intended to replace the existing Launch Processing System (LPS).

The GCS system provides the following:

- Will replace obsolete, difficult to maintain hardware and software.
- A baseline which supports multiple applications (shuttle and space station).
- Will be heavily based on hardware and software standards.
- Allows reconfiguration of hardware and software for testing purposes (subsets).

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

The TPOCC consists of eight software subsystems:

- Operator command.
- Operator display.
- Events.
- GMT.
- Spacecraft communications.
- Spacecraft telemetry and command.
- Spacecraft communications.
- Spacecraft telemetry and command simulation.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

TPOCC is a new system under development at Goddard Space Flight Center. TPOCC is intended to replace the existing MSOCC system.

TPOCC provides the following:

- A baseline for support for multiple satellite missions.
- Can be tailored to support satellite-specific processing requirements.
- Is based on industry standards, including UNIX, C, TCP/IP, UDP, RPC, XDR, NFS, and X Windows.
- Will replace existing, outdated hardware and software.

TPOCC is currently in the prototype stage (PTPOCC). The PTPOCC will be used in conjunction with the existing system to provide proof of concept.

The central concept of TPOCC is use of portable, modular subsystems which use a network transparent means of subsystem to subsystem communications. This allows the hardware to be tailored based on the processing requirements of the mission.

Survey - Executive Systems Introduction

The systems surveyed include several operational and planned telemetry processing/executive systems in use at different NASA sites:

- Transportable Payload Operations Control Center (TPOCC).
- Generic Checkout System (GCS).
- Multi-satellite Support Operations Control Center (MSOCC) Application Executive (MAE).
- Space Flight Operations Center (SFOC).
- Image Reduction and Analysis Facility (IRAF).
- Peripheral Processor System (PPS).
- Trajectory Operations Application Support Task (TOAST).

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Some of the features present in SVR4 include:

- From System V: the interprocess communications facilities, an improved implementation of streams, the Transport Level Interface (TLI), and the Remote File System (RFS).
- From BSD: the C shell, the Berkeley fast file system, selected commands and system calls, symbolic links, sockets, inetd, and the r* commands.
- From SunOS: memory mapped files, dynamic linking, shared libraries, NFS, RPC, and XDR.
- New features including: internationalization, the Virtual File System, the Korn shell, new mechanisms/interfaces for network selection, name to address mapping, and mail.

System V Release 4 (SVR4) is expected to become the industry standard UNIX variant operating system.

SVR4 offers the following advantages:

- Combination of System V Release 3, BSD 4.3, and Xenix.
- Fully compatible with SVR3, BSD4.3, and Xenix.
- POSIX compliant.
- Offers real-time capability.
- Available in first/second quarter of 1990.
- Expected to be reliable and robust.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Although the X protocol is proceeding toward standardization, the windowed applications created with X have different user interfaces. The objectives of this committee are to develop:

- An Application Programming Interface (API) standard for the toolkit layer of X to provide procedural interfaces for common services which will allow application portability.
- A recommended practice for drivability of user interfaces which will allow users to move easily between applications and systems from different vendors.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

ANSI C status:

- Technical issues resolved.
- Procedural issues still not resolved.
- ISO adoption.

ANSI C provides:

- Standardization of libraries.
- Function prototypes.
- Superset of K&R C.
- Does not provide multitasking support.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Networking Standards (1003.8) have not completed their specification. The end result of their work will probably include:

- Message Handling System (X.400)
- File transfer and Management (FTAM)
- Association Control Service Element (ASCE)
- Directory Services (X.500)

Real-time Extensions include:

- Binary semaphores
- Process memory locking
- Shared Memory
- Priority Scheduling
- Asynchronous Event Notification
- Timers
- IPC Message Passing
- Synchronous and Asynchronous I/O
- Real-time Files
- Threads

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Standards
POSIX Security

The POSIX security standard does not mandate a specific security policy model. It defines functions and commands which may be used to support a number of types of security policies, including nondisclosure, integrity, and accountability. Currently, the standard defines the functions and commands which will support:

- Discretionary Access Control
- Mandatory Access Control
- Security Auditing
- Least Privilege
- Object Reuse
- Object Import and Export.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

This standard will be broken down into two parts, a section defining core requirements independent of any programming language, and a section composed of programming language bindings. The topics currently identified are:

- Global Concepts
- Environment interfaces provided to applications.
- C language interface provided to applications.
- Shell command line interpreter language.
- Execution environment utilities.
- Application installation environment utilities.
- Optional software development environment utilities.
- Optional C development environment utilities.
- Optional FORTRAN development environment utilities.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Standards
POSIX Shell/Appl. Utility Interface

The goal of this working group is to specify a standard source code level interface to shell services and a common utility program for application programs. It will include:

- Program primitives to specify shell instructions.
- A standard command language for the shell.
- Recommended syntax for utility naming and argument specification.
- Primitives to assist in parsing and interpreting utility arguments.
- Recommended environment variables.
- A minimum directory hierarchy.
- Utilities to perform common tasks.
- Optional utilities for software development.
- Utilities and standards for the installation of applications.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

User Interface Management Systems
OSF Motif

Motif is a UNIX user interface standard for creating a consistent and easy-to-use graphical user interface on systems from multiple vendors. It provides:

- Toolkit - a varied collection of widgets and gadgets for building applications and a standard graphical interface.
- Window manager (MWM) - manages the operation of windows on the screen with window management facilities and provides an industry standard user interface with a high degree of flexibility.
- User Interface Language (UIL) - presentation description language which allows developers to create simple text files which describe the visual properties and initial states of interface components.
- Style Guide - provides guidelines for using toolkit widgets, designing new widgets, and customizing window managers.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

The Transportable Applications Environment is a portable, integrated environment for designing and building user interfaces. TAE includes:

- TAE Plus Workbench - development tool for the interactive design and layout of user interfaces.
- Window Programming Tools (WPT) - package of application-callable routines used to display and control a user interface.
- TAE Plus Terminal Monitor (TM) - an applications executive which provides user functions in both command line and menu-based user modes.
- WPT TAE Command Language (TCL) - a set of interpreted WPT commands which allow the developer to create rapid prototypes of graphical applications.
- Facelift - allows ASCII-based interfaces to be changed to a graphical look and feel.

NASA Grant No. NAG 9-340
SwRI Project No. 05-2881

Abstract

In order to satisfy the diverse graphics requirements of the MCCU, it will be necessary to utilize several different graphics software tools. The key will be to use tools which are portable, compatible with X Windows, and best suited to the requirements of the associated application. From a high-level viewpoint, this will include a User Interface Language (UIL), an interactive display builder, and a graphic plotting/modeling system such as GKS or PHIGS.

1.0 Introduction

The graphics requirements of the Mission Control Center Upgrade (MCCU) environment are quite diverse. The requirements range from simple text display to user interaction to complex 2 and 3-dimensional modeling. These requirements are most effectively satisfied with a combination of graphics software tools, each utilized for those applications in which it is most appropriate.

Satisfying a diverse set of requirements and at the same time using standard graphics software, has been a difficult task in the past. If portability was required, requirements were often met by utilizing graphics software tools which were not ideally suited for the application. For example, it is not appropriate to use a high-level graphics system such as GKS or PHIGS to display text or present user interface objects (such as colored rectangles which change orientation when selected).

In the near future, all levels of standard graphics software will be available. This software is already available, but in many cases is public domain and is unsupported. When all software is available as supported products, it will be possible to meet all graphics requirements of the MCCU with standard tools which are efficient for development and execution.

2.0 X Windows and Graphics

The foundation of all graphics generated in the MCCU will be the X Windows system. In the near future, all graphics requests will ultimately go through the X Windows system, thus allowing all graphic input and output to be transparently shared across the network. The X Windows system consists of a client-server model in which every display is managed by an X Windows server. A client communicates display requests to the server (which may or may not be on the same system) via a standard protocol (the X protocol). Every graphics request, no matter at what level it was initiated, will ultimately be translated into the appropriate X protocols and routed to a server for display. This routing from client to server may be local or across the network, thus allowing graphics to be generated on a host system (possibly a data or compute server) and then displayed on any workstation in the network.

One problem with the X Windows system is that benefits of the standard protocol are only realized when using graphics over a network. For local graphics interaction, the translation and routing of the X protocol reduces the performance of graphics. Many vendors are addressing this problem by using shared memory or some other fast medium for routing of client-server protocol. This increases performance, as a medium such as shared memory is significantly faster than typical network communication methods (TCP, ISO). However, overhead still remains due to the protocol translations.

A more serious problem is that the X protocol currently supports only 2-dimensional graphic functionality. Using 3-dimensional functions is difficult as they do not closely translate to available X Windows protocols. To solve this problem, vendors of 3-dimensional software (such as PHIGS) will cooperate with the X Windows system (work within its windows), but actually bypass the cli-

ent-server routing and directly access the native system graphics. This allows local use of such graphics software, but precludes its use over the network. Fortunately, a solution to this problem is in the works in the form of standard extensions to the X protocol. These extensions are called PEX (PHIGS Extensions to X). They implement 3-D functionality in the X protocol, thus allowing 3-D graphics requests to be efficiently processed.

3.0 Contents of the X Windows System

X Windows is a relatively new standard and as such, does not provide a complete set of the desired (or required) graphics development tools. As defined by the X Consortium, an X Windows system as provided by a vendor must provide the following applications and libraries:

- X Windows server,
- XLib library, and
- Xt library.

This combination of applications and libraries is not adequate to efficiently support all graphics requirements. The Xlib function calls operate on a very low level and are quite difficult to use. The Xt function calls define a standard whereby user interface objects (called "widgets") may be defined and manipulated. It does not include any widgets or a higher level library for presentation of user interfaces.

Note that another requirement for effective use of the X Windows system is a small set of clients, including a terminal emulator, window manager, environment manipulation applications, and other applications. Such clients are normally provided with an X Windows system, but are not part of the standard as defined by the X Consortium.

To provide a complete set of graphics development tools, additional libraries and applications must be layered upon the Xlib and Xt libraries. These include the following:

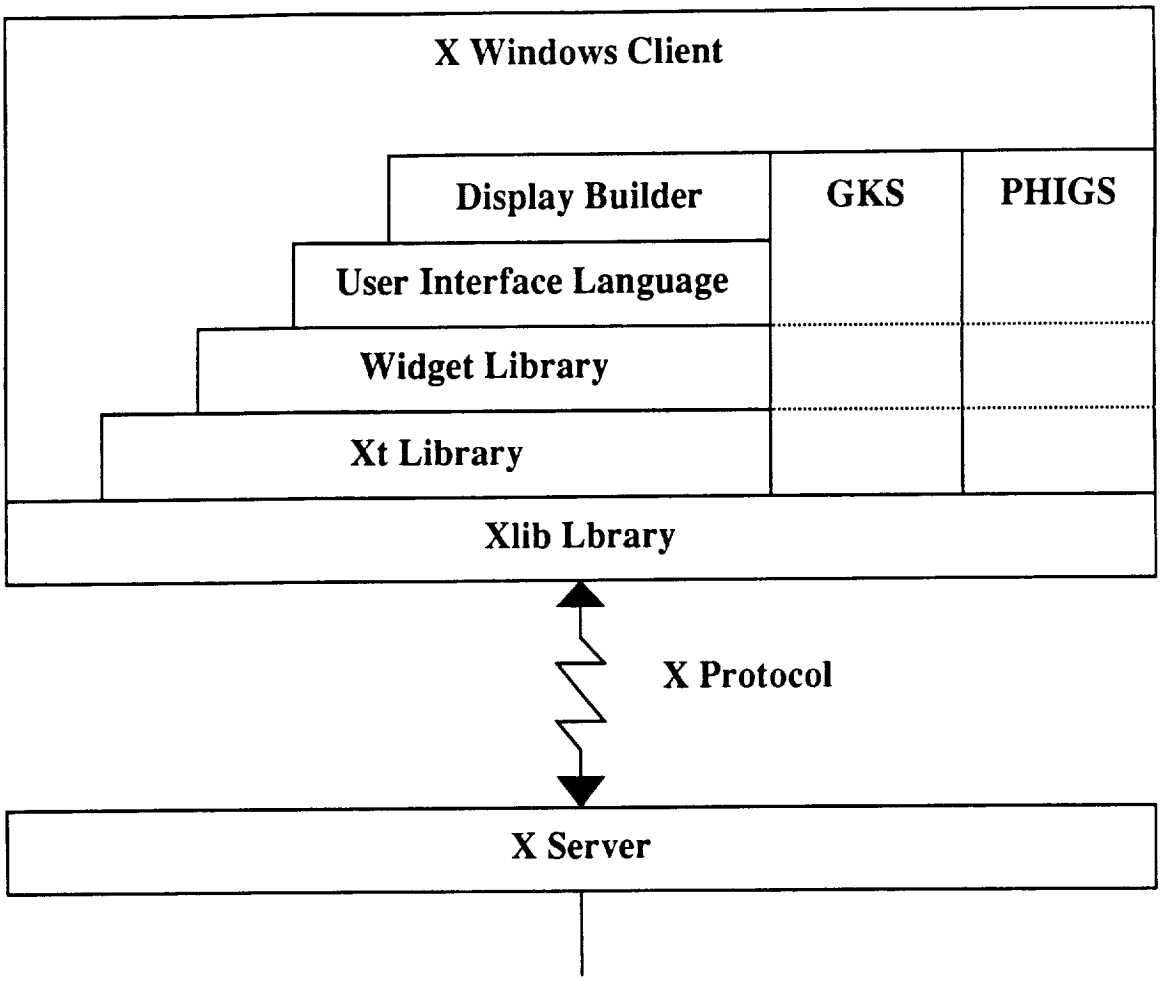
- A library of widgets,
- a User Interface Language (UIL),
- an interactive display builder, and
- a high-level 2 or 3-dimensional plotting/modeling system.

The entire X Windows system including the server, basic libraries, and high-level tools is graphically summarized in Figure 1.

Most users in the MCCU environment are familiar with the term widgets. Widgets are individual user interface objects such as command buttons, menus, sliders, etc. Using the Xt library and a set of widgets greatly simplifies presentation of user interfaces. Widgets operate at a high-level and automatically handle typical window system requirements such as cursor redefinition, resizing, and highlighting of active objects.

Although widgets may be directly used by the programmer, it will be more efficient to utilize a UIL to define a client's display interface. A UIL allows a programmer to completely define a user interface without actually developing or modifying code. This provides the following advantages:

- Significantly reduces development time,
- allows user interfaces to be rapidly prototyped,
- allows simple and global updates to be rapidly implemented, and



Note: GKS and PHIGS provide their own user interface objects for the various logical input devices supported. Ideally, the GKS and PHIGS implementations will utilize widgets and the Xt library to support such functions. This of course will be implementation-dependent.

Figure 1 - X Windows Software Structure

- separates user interface definition from actual code (important for CM).

Note that the final advantage is important as it will allow changes in the user interface to be implemented without affecting the actual code. The UIL description would still be certified, but this would be simpler than recertifying an entire application.

Although a programmer can design a user interface by "coding" in the UIL, it is more efficient to utilize a tool which allows the interface to be interactively designed. This display builder will allow the programmer to interactively select and place user interface objects (widgets) on a blank form. The programmer is thus able to design and arrange the interface and receive immediate visual feedback. The end result of this process is a file containing the corresponding statements in the user interface language.

The final applicable graphics tools include 2 and 3-dimensional plotting and modeling systems. This includes GKS (Graphic Kernel System) and PHIGS (Programmers Hierarchal Interactive Graphics System). From a superficial standpoint, the functionality provided by these systems is the same as the low-level X Windows libraries. However, upon closer examination, these systems have a great deal more functionality. The primary advantage of these systems is that they allow objects (whether simple plots or complex models) to be defined and then scaled, rotated, and transformed in other manners. For example, a 2-D model could be scaled to simulate it moving closer to or farther from the viewer. A 3-D model could be rotated to view other sides or to view affects of light on its different surfaces. Whereas in X Windows, integer device coordinates are used, GKS and PHIGS allow floating point coordinates, the ranges of which are defined by the programmer. The transformations from these coordinates to the actual device coordinates require a large number of floating point calculations. This amount of floating point processing makes the overhead of using GKS or PHIGS too great for simple drawing applications, such as displaying text, drawing lines, or presenting a user interface.

4.0 Meeting MCCU Graphics Requirements

Although the graphics requirements of the MCCU are quite diverse, they may be summarized into a few categories. These include the following:

- Window-based user interface,
- high-performance, high-flexibility data-driven display, and
- plotting and modeling.

The key to meeting MCCU graphics requirements is to use the appropriate graphics software for each application. Providing a window-based user interface is achieved by using the Xt library and a set of widgets. Layered on top of this will be the UIL and the interactive display builder which utilizes it. These tools will be used by all applications which present a graphic user interface. This includes system applications (such as WEX) as well as user applications.

The display builder application (and underlying UIL) must be flexible enough to support all MCCU requirements. This not only includes development of displays and placement of user interface objects, but also a way for users to associate their data with output objects (widgets). For example, the user may require some subset of data (real-time or other) to be displayed at a defined interval. The output may consist of text and/ or objects which graphically display results (such as a dial, a gauge, etc). The widgets provided for this purpose must balance performance with visual and graphic flexibility. For example, a widget must be provided to display and update a large number of data values (>100) in a manner which does not unduly affect the performance of the system.

Other widgets will be provided for less dynamic or volumous data, in which more graphic presentation is desired (it is easier to recognize a critical condition by examining a graphic dial which is approaching a marked level).

The combination of user interface and data demands make it difficult for any existing graphics tool to satisfy the requirements. What is required is a general-purpose UIL and display builder which allows development of client displays containing the required set of user interface widgets, a number of which may be automatically driven by external data. Some of the existing or planned systems which support a subset of this functionality include the following:

- Display Builder application developed by Ford Aerospace,
- the Data-Views drawing system,
- the Open Software Foundations (OSF) MOTIF system, and
- the Transportable Applications Environment (TAE)

Each of these systems has its advantages and disadvantages. SwRI's recommendation is to use one of the systems (or an equivalent) and either modify it to provide all required functionality or integrate it with other products to achieve the same goal. The end product of such a system should be a language which is used by actual code to present the user interface and drive the display of data.

The UIL and display builder will not satisfy all MCCU requirements. There will remain applications which must construct complicated graphic plots and/or models. For such applications, if the requirements are too complex for one of the X Windows software tools, a system such as GKS or PHIGS should be used.

There are two basic approaches for integrating GKS or PHIGS graphics into an X Windows application. The first is for the application to exclusively use GKS or PHIGS and use their inherent capabilities for user interface. As described in Figure 1, GKS and PHIGS provide their own user interface functionality. However, using this approach will most likely (depending on the implementation) make the applications interface different from others in the environment. The second approach would be to use the display builder to design a screen which includes a blank "canvas" area, which may be as large or small as is desired by the programmer. This area will be used as a virtual workstation into which the GKS or PHIGS output may be directed. In this approach, X Windows functions will be used for user interface requirements. The second approach is preferable, but requires additional functionality in the UIL and display builder.

In SwRI's understanding, the requirements for GKS or PHIGS are not common in the MCCU environment. This however may change in the future and it does not make sense to design other software which will preclude the use of these graphics tools. If the other graphics tools meet all requirements, only a small subset of users will actually require use of GKS or PHIGS. Such instances should always be reviewed to determine if the software is really necessary.

Note that there remains a decision as to whether GKS or PHIGS will be used for the related requirements. Although the packages are conceptually similar, there are some major differences. GKS is currently a more stable graphics standard. It is supported by more vendors and is significantly faster than PHIGS. It provides adequate 2-dimensional capability for static display of graphics. On the other hand, PHIGS is a relatively new standard and will by its very nature, be slower than GKS. However, PHIGS provides 3-dimensional capability and more importantly, allows dynamic changes to images. When using GKS or PHIGS, a programmer will normally create a seg-

ment (GKS) or structure (PHIGS) which defines some graphics object. In GKS, if that segment is to be changed, it must be destroyed and then completely recreated with the required changes. In PHIGS, it is possible to specify only the changes, thus making it more efficient for very dynamic images.

A final advantage of PHIGS is that its described functionality will be added as an extension to the X Windows protocol. PEX (PHIGS Extensions to X) will allow PHIGS graphics requests to be directly implemented in the X protocol, thus improving performance and insuring that the graphics will work over a network. In the case of GKS, the requests must be converted to the appropriate X Windows protocol. Essentially, GKS is a good short-term solution for applications which will not require 3-D or dynamic image functionality. PHIGS however, for the reasons outlined above, is a better long-term solution.

1.0 Overview

This document describes any problems (potential or otherwise) seen with IBM's Workstation Executive (WEX) 2.5 preliminary design. The PDR document was divided into 12 sections and several appendices. This document includes comments on the sections describing workstation software. This includes the following:

Section #	Section Title
1	System Overview
2	GPLAN/RTLAN
3	WEX
7	Data Acquisition
9	Configuration Management
11	2.5 PDR Systems Analysis

Note that no comments were generated for section 12 (CM Workstation), as this is a prototype for delivery 2.5. The described design appears to be sound, however there are details which must be completed before this application can provide operational support.

The following sections of this document include general comments and then specific comments for each applicable section. For sections providing specific comments, the page in the PDR document is given and then is followed by the appropriate discussion.

General Comments

In general, the design presented by IBM is sound. Many of the concepts demonstrated by the Hardware Independent Software Development Environment (HISDE) prototype were reflected in the 2.5 design. From the 2.5 PDR, it is obvious that IBM is migrating towards a hardware independent design. This is evident in the use of X Windows as the primary graphics software. In recent versions of WEX, one of the primary problems is the dependence on proprietary graphics systems (namely MGI graphics). By removing dependence on this software and forcing users to do the same is a major step toward writing portable code, as the majority of non-standard user software is in the area of graphics use.

The 2.5 design also includes use of ISO standards as implemented by the RETIX corporation. These ISO standards are based on the Open Systems Interconnection (OSI) reference model and is an important step toward standard usage and communication over LANs. However, it is important for IBM to recognize that such a move will initially cause problems due to existing software which is based on other standards (namely TCP/IP). TCP/IP is relatively fast, reliable, and has a wealth of application software which relies upon it. During the transition from TCP/IP to ISO, some means of retaining these benefits must be retained. Users and developers will not want to lose features such as network X Windows, NFS, EMAIL, and others, which currently work well over TCP/IP.

There is concern about IBM's understanding of what is SVID UNIX and how it will be verified that their software is truely compliant. IBM should not state that their software is compliant when it allows or depends on functions which are unique to Berkeley UNIX. IBM's design should allow the user of non-SVID commands and features, but should constantly remind users that such functions may not be available on other systems. Also, IBM's design should in no way depend on any non-SVID commands or functions.

The primary concern about the 2.5 design is that it is not completely hardware independent. Although IBM states that the design is in fact hardware independent, there is no provision whereby this will be proven (such as porting software to another workstation system). In addition, the basic design is still dependent on a tightly-coupled processing configuration, in which there is no distribution of data or processing. All computation (less that on the MASSCOMP IGP's) and data is resident on the main CPU(s). This design is not flexible enough to support a true diskless node configuration, in which each node has a significant amount of local processing power. The diskless node approach is one chosen by many of the industry workstation vendors and cannot be ignored if one is attempting to produce a hardware, vendor, and configuration independent system. This is a difficult problem and one in which there are many opinions but only a few reasonable solutions.

2.0 System Overview Comments

Page 1-10

The document states that the 2.5 design is hardware independent. Again, this is not true, as the design is neither vendor nor configuration independent. If the design is truely hardware independent, then some plan to prove this should be devised.

3.0 GPLAN/RTLAN Comments

Page 2-4

Their is no reference to how existing TCP/IP-based services will be provided in the interim before they are supported via ISO mechanisms. This will preclude use of services such as network X Windows, NFS, UNIX EMAIL, etc.

Page 2-5

There is no indication of how many of the ISO layers are implemented in the UNIX kernel. From previous discussions, layers 3 - 7 operate at the user level. This will may make communications extremely slow. IBM should be requested to provide benchmarks which compare movement of data through existing and then the ISO communications routes. Note that it is unlikely that ISO will ever be faster than existing methods, but there are steps that can be taken to improve performance (such as placing more ISO software into the UNIX kernel).

Page 2-14

Why is it necessary for the user to format data into 8K blocks? While it is understandable to maintain this functionality to be compatible with WEX 2.3, why not provide a new utility which allows access in a more UNIX-like fashion?

Page 2-15

Is it true that LAN services do not require WEX shared memory? The shared memory segment allocated by WEX is quite large (about 0.5 to 1.0 MEG). It is unreasonable for a user to lose resources to WEX shared memory if only LAN access is required. Separating the two services is an excellent design idea, but care should be taken to avoid any dependencies between the two software systems (such as LAN services requiring WEX shared memory services to determine flight or operation mode).

Page 2-26

What is the purpose of "network parameters"? If these are upper limits, what will happen to data which exceeds the limits?

Page 2-31

Is the data saved by the network manager available for local display to the user? Also, is there some sort of client which allows this data to be displayed in an interactive and meaningful manner? Finally, can this data be routed to the Health and Status application?

Page 2-32

Will the file transfer and access capabilities of FTAM allow use of wildcards and recursive access of directories? Users will require the ability to copy the contents of entire directories and hierarchies. This capability will also be required for WEX applications which move files to and from the host and other workstations. If FTAM itself does not provide these capabilities, a set of shell functions must be provided. These functions must be available for interactive and programmatic usage.

There is no mention of how IBM's FTAM will honor normal UNIX file ownership. From experience, FTAM does not do an adequate job with file ownership. The ability to entirely enable/disable workstation access is insufficient. Users will require permissions to be recognized on a per file basis (just like existing UNIX services provide).

Page 2-37

FTAM's honoring of the UNIX "other" ownership is inadequate. This precludes recognition of user and group ownership.

Marking a file as "non-executable" is a good idea, but is by no means fail-safe. If the user is able to write the file, he will be able to change the permissions on it, thus quickly making it executable. If FTAM is the sole means of copying files over the network, it could be used to determine if a file is truely executable (via examining magic numbers) and if so, prevent it from being copied. This would be a reasonable location to place such security checks.

Page 2-43

The network test tools should be available for programmatic access. This would allow an application to query the host and/or workstation before access.

4.0 WEX Comments

Page 3-4

WEX does not provide "vendor independent" applications software. If this is a primary goal, IBM should modify their basic design to support other vendors and configurations.

IBM states that the user may use alternate shells, window managers, terminal emulators, and other critical tools. This is acceptable during development mode, but during operational mode, such applications must not be allowed unless they are certified. It also makes sense to only allow use of standard applications during operational mode. This would further force users to develop portable code and not depend on non-standard tools.

Page 3-5

WEX does not adhere to SVID UNIX. It depends on and allows Berkeley-specific tools (for example, the document describes two startup sequences, one of which is unique to Berkeley UNIX). It also does not specify any means whereby SVID compliance is to be verified.

ANSI C extensions should not be used until they become more universally accepted by workstation vendors.

Page 3-6

Stating that WEX applications "will run with or without the window manager" involves a significant amount of functionality. This means that if a window is resized (larger or smaller), the client will intelligently resize areas and fonts. This is not an automatic process (at least not "intelligently"), as most "widgets" do a poor job of handling resize operations.

Page 3-7

WEX must not preclude the use of PHIGS. The current architecture, if used with true X terminals or diskless nodes (treated as X terminals), may preclude use of PHIGS if the software bypasses the X server. At the current time, much high-performance graphics software bypasses the X server. The graphics software cooperates with the window system, but does not route display requests through the X server. Instead, the underlying graphics libraries are directly accessed. This is primarily due to lack of X protocol features for functions such as 3-dimensional rendering. In the future, the X protocol will be expanded to support such functionality, but in the interim, such graphics will not use the X server and therefore are not network transparent.

Mouseless mode is an interesting problem which may be solvable. It is definitely solvable with access to server source code.

Page 3-8

WEX should provide a user interface language (UIL). This includes some sort of high-level language for definition and separation of user interfaces. This language is accessed programmatically and/or through an interactive display builder (such as that provided by TAE). Note that the Display Builder application (as defined by FAC), is not general purpose enough to support this requirement. A UIL would drastically reduce the amount of time required to develop WEX and application user interfaces. Also, as the user interface specifics are separated from the actual applications, it makes it very easy to alter the interface.

Page 3-12

WEX does not support diskless nodes as separate processors. It only allows them to be used as dedicated X Windows terminals. This would waste the local processing capability of the diskless nodes.

WEX will not be capable of supporting X terminals. Such devices have minimal graphics support and rely heavily on the host (the MASSCOMP) for computation. Using X terminals would place an unreasonable burden on the MASSCOMP (a burden which is currently relieved by the dedicated graphics processors in the IGP's).

Page 3-14

If WEX is only going to run on the MASSCOMP 6600's and is SVID compliant, then why are two start-up scenarios presented (one of which is not SVID)?

Page 3-18

Why is WEX shared memory allocated even if the user is not planning to use any WEX service?

Page 3-19

The WEXTASKS file requires a new language. As an alternative, use standard UNIX services to provide the required functionality.

Page 3-22

Is it possible for the user to send an advisory to a user on any given workstation? The document states that an advisory may be sent to any tty, but not another workstation.

Page 3-23

If a user is logging into the system in OPERATIONAL mode, where is the login verification data coming from? Is it the local CM process or from the host? This is most likely from the host, but if local, some sort of extra security is required to prevent unauthorized access to the data.

Page 3-25

In the "xterm" window provided during login, it is not clear what application is running (it is normally the shell). This is of course not reasonable during login. Also, the xterm window should have a scrollbar to allow the user to review messages which have scrolled off the top of the screen.

Page 3-26

The document references "keyboard-to-button" mappings, which are assumed to be inputs such as tab and cursor keys. Such mappings MUST be consistent from WEX application to application. These mappings must be sufficient to allow a user to complete a screen without ANY mouse interaction. It is very frustrating to be forced to use a combination of a mouse and keyboard interface.

The keyboard mappings must also be provided to users so that they may make their own applications consistent with those in WEX.

Visually disabling buttons or fields is in many cases a costly and unnecessary action. It is normally adequate to use a simple mechanism such as preventing the cursor from changing when it tracks over a button. The best solution will depend on the user interface tools used (which widgets) and how efficiently they handle such requests.

Will the user be allowed to selectively kill background processes, or will it be a "all-or-nothing" action?

Starting up the users environment in the xterm window will be a problem, as some users will not want to present an xterm window to their flight controllers. If users want an xterm window available, have them specify it in the "xstuff" start-up file. Give the user a completely blank starting point from which they may set up the environment.

Page 3-30

If a download or recycle is requested, will users on other terminals be able to prevent it from occurring? If one user wants to enter a mode requiring such action, will it cause users in other modes to be logged off?

Page 3-31

Why are UNIX accounting files updated? Is this for security reasons? Using the UNIX accounting process will use a significant amount of resources.

Page 3-36

All common WEX default resources (colors, fonts, etc) should be stored in the server so that clients may be quickly initialized. Otherwise, the client will have to read and parse such data from a file each time it is executed. Note that storing defaults into the server is made possible with the "xrdb" client.

Page 3-41

Is the "icon" button on every title bar really necessary? If the reasoning is to account for situations in which no window manager is running, then buttons for move and resize should be considered. It is not a good design to have application-specific buttons for such common functions. This type of functionality is provided by other existing window managers. If IBM prefers a window manager which places move, resize, iconify, and other widgets on each window, then one providing these features should be selected and agreed upon.

Again, how intelligent will the resizing action be? On many clients, the user will only want to resize a portion of the window, such as that used for data entry or output (such as the message area of the advisory client). Any client which uses a large area for display of a variable amount of information should initialize separate windows (Xtoolkit shells). This allows the user to easily adjust the amount of text displayed.

Note that many simple messages (especially those requiring immediate response), are best processed with popup windows.

All important messages should be output to the advisory window if it is currently active. Note that local message windows are not useful if the advisory window is already present. They are only really useful if large amounts of data must be displayed (such large amounts of data would clog the advisory window).

Page 3-44

The document makes frequent reference to on-line help, but there is no sample screen nor any real definition of the functionality to be provided. Will it be context-sensitive (more ideal) or more of a window with "man" formatted data? An ideal solution would be to provide both types of help.

Is there a process which handles on-line help or is this a sub-function of all clients? It makes more sense to separate this function, as it will make clients easier to develop. It also will allow clients to run in parallel with the user obtaining help (without forcing the client to worry about retaining parallel operation). Note that this does make context-sensitive help more difficult to implement.

Will the local CM access functions be converted to X Windows? The windows shown appear to be in a "curses" type interface.

What type of display interface is being used by the applications which are not being converted to X Windows? WEX cannot claim a consistent "look and feel" if there are applications which use a different interface. Also, are these interfaces dumb screen I/O, curses, or MASSCOMP graphics? All should be redone, but especially the MASSCOMP graphics, as this type of code is completely non-portable.

Page 3-47

As a general comment, on prompts which request entry of a number in a limited range, it is often easier to use a scrollbar to adjust the value. The best solution is to provide a text area in which data may be entered directly and some sort of widget which allows mouse-based manipulation. Using up/down buttons (as shown in the example) are inefficient for wide ranges of data in which precision is still required.

Page 3-48

How does the user acknowledge a message in the advisory client's window? What does the "browse" function do? It should provide additional functionality such as search for text, display on message type, sort, etc. The browse function itself should be integrated with the main window functions (make window bigger to browse).

Page 3-54

What does the "registration" process involve? This implies that all WEX shared memory is already allocated and the registration identifies the user to WEX, attaches the user to shared memory, and provides some resources (which the user may or may not need). It is not reasonable to allocate a large amount of shared memory unless the user uses the associated WEX services. Shared memory and other resources should be allocated dynamically as required by the WEX services used. If no WEX service is used, NO shared memory, message queues, semaphores, or other resources should be allocated. A typical user may only require data acquisition services and should not lose resources due to requirements of undesired services and applications.

Page 3-55

Most users sophisticated enough to use "message queue utilities", "interprocess communication utilities", and "shared memory buffer utilities" will neither require nor desire automatic initialization. Again, all WEX resources should be dynamically allocated and only when requested by the user. Why should a user lose a large amount of memory if he only needs to use a simple service (such as outputting an advisory message).

Page 3-56

What is the provision for returning resources after an application terminates unexpectedly (without calling EXwexterm)?

Page 3-62

Why is the "su" command provided? While a user may want to "su" to another user (other than root), it still implies that the users will have root access. If this is so, there cannot be any reasonable CM, no matter how stringent the policies are during operational mode (short of downloading the entire system).

Page 3-63

What is "proper security"? Is this a dependence on procedural control? Allowing a user to read in data allows import of executable files. It may be necessary to "front-end" any and all such commands.

There are a number of commands listed which could be dangerous during operational modes. For this scheme to work effectively, strict permissions must be maintained on all files.

5.0 Data Acquisition

Page 7-9

The user interface presented is inefficient in its use of display space. The areas for "Active MTM streams", "Active Positions", and pop-ups should not be static. It is better design to have separate windows which are popped-up as required (in the same manner described for local message windows).

Page 7-14

Why is the Uniform Network Interface not used for both LAN's (GP and RT)? Will the user be allowed to access the GP LAN via this set of tools?

Page 7-15

Is it guaranteed that the WEX shared memory is not required for LAN access? It appears that WEX shared memory is required to log in, so there is nothing gained by separating the resources used by the two services (other than one allocation being static while the other is dynamic).

6.0 Configuration Management

Page 9-6

Will the "Workstation Processor ID" adequately differentiate between different types of processors for a given vendor (MC5500 vs MC6600)?

An element will need to indicate for which release of the operating system it has been loaded for. It will also need to indicate the release of other major software, such as graphics, WEX, X, etc.

Page 9-12

How is the "integrity check" performed? What is the baseline against which data is compared? Is it simply update dates or some sort of checksum? Either way the baseline data must be resident on the host or protected from user access.

Is it possible for users to modify the permissions (make executable) on files in the operational area? If so, the user could easily make executable, a file which appears to be a normal binary data file.

Page 9-13

What does "linked into the operational area" imply? Does this mean (in the UNIX sense) that links are established to existing files somewhere else on the file system?

Are all executables downloaded or just those not already on the disk? If an integrity check is performed (and the check is dependable), is it necessary to download files which already exist? A complete download makes sense from a security standpoint, but will take a significant amount of time to complete.

Page 9-14

The described file tree indicates that the "chroot" command will be used. In such a scenario, many of the common areas (usr, bin, dev, etc) will have to be duplicated. Again, this makes sense from a security standpoint, but will waste disk for areas which are identical for all operational modes.

Page 9-26

The element information includes the "setuid" permission. Is it possible to set this bit and set application ownership to root, thus allowing the application to access the system as if it were superuser?

Page 9-38

How will "reads of executables" be prevented during operational mode? It appears that the "cpio" and "tar" commands will be available and they allow such operations.

Page 9-39

Is it likely that a user would ever initiate an integrity check? This might be used if the user is encountering a strange problem which he suspects is due to out-of-date code or system data. In such a case, a more useful feature might be a simple check which examines the revisions of OS, WEX, graphics, and applications.

Page 9-41

Again, how is an "uncertified" program found? How dependable is the method used to determine if a file is an executable? Is it simple permissions, magic numbers, or actual file content?

Page 9-44

This screen shown for CM does not indicate that it will be ported to X Windows. Does this mean that an existing curses (or something else) interface will be retained?

7.0 2.5 PDR Systems Analysis

Page 11-6

The cumulative MIPS figures indicate that the Flight Support Host will be used near its processing capacity. Expecting a system to have adequate performance and response after 75% load is unreasonable.

Page 11-20

Which release of X Windows was used for the benchmarks? MASSCOMP has released version 1.1, which is significantly faster than release 1.0 (Note that both are based on the MIT X11 Release 2 X Windows).

The table indicates that the Display Manager uses X Windows. Is this really the case, or will this application still use GKS as its primary graphics resource? Note that if it uses GKS, it may be slower than X Windows (depending on the type of fonts displayed).

Page 11-24

The document states that "6.4 MIPS" will be provided for user applications. This assumes that the machine is capable of efficiently providing 100% of its maximum rating processing capability. This is unreasonable, as when more and more processes are executed, an increased amount of processing is lost to system overhead; therefore the "6.4 MIPS" number is optimistic.

1.0 Overview

This document describes any concerns (potential or otherwise) seen with applicable portions of the 2.5 critical design. This document concentrates on sections 5 and 6, which deal with WEX and local Configuration Management.

2.0 WEX Comments

Page 5-6

The term "vendor independent" is misleading. Although software is based on standards, it depends on a configuration which is quite unique within the workstation market (that being MASSCOMP's).

The document states that "WEX services will be provided on an optional basis". Is this absolutely true or do major applications like data acquisition and configuration management require initialization of WEX before they will operate properly?

Providing a "uniform look and feel" encompasses a broad area of user interface. For this statement to be accurate, a well defined standard for colors, menus, title bars, widget use and placement, keyboard mapping and other aspects must be defined. A functional standard for the user interface should be designed, presented in the form of a prototype, and used for all applications executing on the workstation. Note: will applications from other contractors have a different appearance? This will be a problem for users.

How significant is the requirement for ASCII terminal support? Will all WEX clients also provide ASCII (curses) interfaces or is ASCII support primarily intended to insure that dump terminals are controlled by WEX? If users are to work from an ASCII terminal, full support from clients is required.

Page 5-7

The document states that SVID UNIX will be used. How will compliance to SVID be verified?

The document makes the statement: "single software architecture will be used across multiple hardware platforms". What platforms (other than the MASSCOMP) will the software be executed and used on? What are the plans for demonstrating this capability?

The "tightly-coupled, multi-processor, shared memory" approach requires a specific, somewhat non-standard workstation configuration. The only other configuration which this design supports is use of workstations or X terminals as dedicated displays. However, even this configuration is in jeopardy, as TCP/IP (upon which X depends) is not available on the GP LAN. Also, has it been demonstrated that the shared memory approach is the only one which is feasible? Has it been proven that a distributed concept using the network would in fact be too slow?

Page 5-8

Using the "uwm" window manager is normally a good choice as it is robust and has little impact on the user environment. Unfortunately, uwm is markedly different from the window manager used by MOTIF (towards which WEX will migrate). Much of the MOTIF "look and feel" is due to its window manager, which places a number of objects (for iconify, resize, move, expose, etc) on every window present on the display. This is the primary means whereby a user manipulates windows, whereas uwm depends on pop-up menus and user-defined "hot-keys" to provide similar

functions. The point is that if the migration is towards MOTIF, it is advisable to select a window manager which eases the transition.

What does the statement "applications designed to run with or without the window manager" imply? It is assumed that this implies addition of functionality to allow windows to intelligently react to asynchronous window events, such as resizing, movement, and exposure. This functionality is expected and is not complete unless each window is able to "intelligently" deal with such events.

Although WEX does not "preclude the use of TAE", it is not understood why a tool like TAE is not used for development of the user interface. This would make prototyping much easier and allow rapid implementation of changes. Although it may prove difficult to learn TAE and port clients to it, this may save time in the long term and improve the quality of the WEX user interface.

Page 5-9

When running an X terminal (or a workstation as such), the server will be resident on the display processor. The statement "WEX is limited to managing X servers on resident workstation" indicates that this configuration would be impossible to use (assuming TCP/IP or some equivalent is available for X protocol communication).

The statement "Majority of WEX/WSA applications are designed/implemented to employ the X Window System" is confusing. Does this imply that some clients will not use an X Windows based interface and will use an ASCII based interface instead?

Page 5-13

What is the timeframe for migration to MOTIF? Will this be part of the 2.5 delivery or will it be included in 2.7? With as much attention as MOTIF receives in this document, it appears that it will be part of 2.5

Page 5-14

Use of the HP widgets is rationalized by referencing use by TAE, yet TAE is not used for WEX user interface development.

The statement "By ensuring that our clients do not have dependences on a particular Window Manager" is incorrect, as presentation of title bars depends on the window manager not presenting its own title bars.

The document states that users will be allowed to use their favorite window manager. This will make the design of WEX clients more difficult as they will have to determine whether or not it is necessary to present title bars.

For more information on these points, refer to the discussion under the heading of "Page 5-52".

Page 5-15

The diagram indicates that 250K of shared memory is required for "WEX buffers". Why is so much memory required? What is the breakdown of memory in this requirement. Is this memory statically allocated?

Page 5-22

Will all WEX clients (and applications) include processing to handle the SIGTERM signals sent during the recycle process? Typically an application includes signal processing code to trap and process such signals. If not present, the application will simply terminate without cleaning up.

Page 5-27

The document states that 60K of shared memory is allocated (in addition to the 250K) for each advisory terminal. Is this memory statically allocated or dynamically as required by the message load? Why is it necessary to save messages in memory? Once messages are displayed, couldn't they be saved in a file? This would save memory and still provide good response for occasional viewing of older messages.

Page 5-33

It appears that the EXitdaemon is required to clean up after users who do not terminate their attachment to WEX. A more effective approach may be to have these checks performed asynchronously (during a EXwexinit call) rather than cyclicly.

Page 5-35

The document states that "the mouse will not be required to login". Will all clients provide equivalent keyboard functionality so that users are not forced to use the mouse?

Page 5-38

Are the Tab/Return and Cnrl/Shift/Return function key mappings available for all WEX clients?

Page 5-43

Why is UNIX accounting initialized? UNIX user and process accounting adds overhead to normal system operation. This function should not be used unless there is a requirement for the accounting information.

Is the user notified of UNIX mail, the ISO X.400 mail, or both? Is the user even allowed to use regular UNIX mail due to the new mail system?

Page 5-45

Why are background processes killed on an "all or nothing basis"? Why is the user not allowed to kill some processes and leave others running?

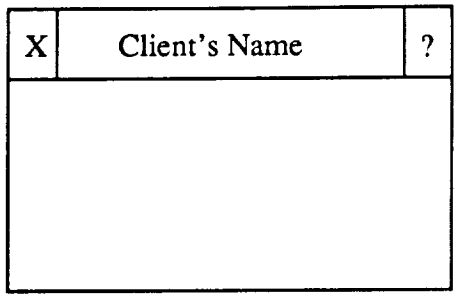
Page 5-50

Note that by changing some defaults, users can dramatically change the operation of their own and WEX clients. For example, if password text is hidden by using the same foreground and background text color, the user can override this and thereby cause the text to be displayed.

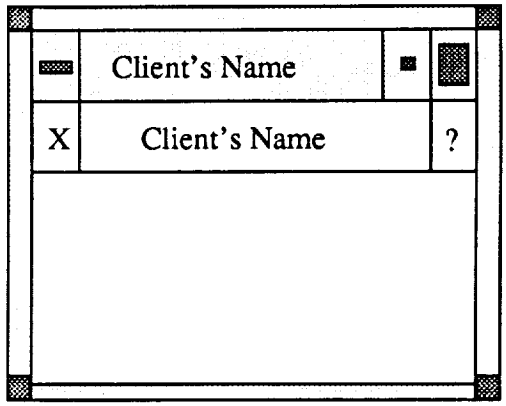
Page 5-52

It is noticed that the WEX user interface includes a standard title bar. This is an excellent idea when using a non-obtrusive window manager such as uwm (or when not using a window manager at all). Unfortunately, this title bar will be redundant when other window managers are used, as they present their own title bars along with a subset of common functions. For example, consider the following example which shows a clients appearance with the uwm window manager and then with the MOTIF window manager:

Appearance (uwm)



Appearance (MOTIF)

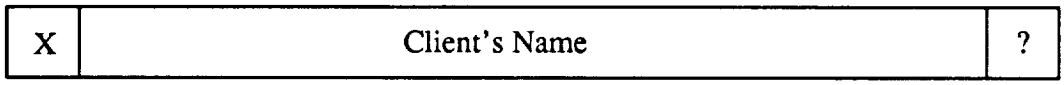


As this example shows, the title bar and some functions become redundant with the MOTIF (and most other common) window managers. A solution is to mandate that all users use one window manager and design the clients accordingly. In this case, you could force all users to use the uwm window manager and present a title bar in each client. Alternatively, you could force all users to use a different window manager (rtl, twm, awm, etc) and assume that the title bar and function responsibilities will be handled by the window manager.

If it is decided that users may use the window manager of choice, then WEX clients should sense the window manager running and present the appropriate interface (whether or not to present a title bar).

In this discussion of title bars, the important point is that all clients have a standard appearance and that common functions are always located in a consistent location. More important than the title bar is the location of common client-specific functions such as terminate and help. It is not necessary for clients to handle operations such as iconify, as this will be handled by the window manager. Rather, a client should present standard functions which are client-specific. It is especially important for an client to present its own terminate function, as using the standard window manager function will often send an unexpected "terminate" or "kill" signal which is not adequately handled.

If all WEX clients are to present a title bar, it is suggested that it appear as shown below:



Where: "X" - Terminate client function
"?" - Help function

In the instances where a title is not required, the terminate and help functions should be presented in a standard location.

Does "application busy clock" indicate that a client is active or process-bound (not available due to processing)? A more common means of achieving this is to change the mouse cursor to an image indicating that the application is busy (hourglass, watch, etc.).

It is not clear what scheme will be used to display local messages. Will each client have its own message area (not recommended) or will pop-up windows be extensively used? For clients with a large amount of output (especially multi-line output), special-purpose output windows are advised. For normal output, pop-up windows are preferred (note that pop-ups must not halt operation of critical applications).

All WEX clients should adopt a standard mechanism for presenting user commands. This could be via a command line at the top of the client from which pull-down menus are accessed. Another option is to place all available commands as buttons along the top or side of the screen. The decision should be based on what users prefer, what is handled well by the HP widgets, and what interface is used by MOTIF.

Page 5-54

The document indicates that several applications do not require changes (no port to X Windows). Is this because the applications have no user interface? If they do have an interface (and are not temporary applications) then they should be ported to X Windows.

Page 5-56

How are the "shift change" and "update rate" pop-up windows selected from the information client?

Page 5-57

If iconified and an advisory is received, will the advisory client become active and display the message? This feature should be available and it should be possible to enable and disable it.

Page 5-61

The WEX Host/Flight list should be a function of the information client. The information client should provide this information as a pop-up window.

Page 5-64

What is the raw format of the help text? Will it be the same as that used for UNIX "man" (nroff with -man macro support) or in straight text format? If in man format, the text could be viewed via the command line. Note that help on clients should be available via the command line. This allows users to access help via familiar utilities.

Is the help text context-sensitive or will the user simply get a large amount of text which must be scanned?

Page 5-71

It appears from this discussion that WEX services are required if the user needs "LAN access utilities". Does this include real-time data acquisition? Some users may require data acquisition but not the full set of WEX services.

3.0 CM Comments

Page 6-11

The document states that files which cannot be linked across file systems will be copied. Files across file systems can always be linked with "symbolic links" (also called soft links).

Page 6-13

The document states that the "chmod" command is front-ended to prevent making a file executable. What protection is used to prevent the programmatic call from performing the same function?

Page 6-29

The user interface for the EXcmmenu client wastes a great deal of screen space. Also, the commands appear to be placed in an awkward location.

Pages 6-31 - 6-36

These pages present several of the windows used by the EXcmmenu client. Note that the user interface for each of the CM sub-functions uses a different approach for presentation of commands.

Page 6-43

The document describes the "replace" option for download via a CDL. Does this option apply to the entire download as a whole or can it be responded to for each individual file?

Page 6-63

Why is no X Windows interface provided for the EXcm_copy and EXcm_erase functions. Also, will any command line interfaces be provided for the functions provided by EXcmmenu?

Page 6-68

The workstation initialization status display is a good idea, but why is the information presented in a special client. At this point, the xterm window (presented at login) should be available for display of these messages.

1.0 Introduction

This section includes two reviews of the proposed IEEE Portable Operating Systems Interface Definition (POSIX) working drafts for real-time (1003.4) and security (1003.6) extensions. This review process involved detailed analysis of the current drafts and comparison with the expected requirements of systems used in the Upgraded Mission Control Center (MCCU) at NASA-JSC.

2.0 Review of POSIX 1003.4

This effort involved a detailed analysis of the proposed POSIX 1003.4 working draft and generation of comments for instances in which the functionality did not satisfy the expected requirements of workstation executive software used in the MCCU. At the time of this analysis, NASA was generating an Operating System Interface Definition (OSID) document. The OSID specifies all interface and functional requirements for the operating systems used for NASA-JSC systems. The OSID primarily consists of a collection of requirements taken from other specification documents (such as POSIX). The comments generated by SwRI were in the form of OSID worksheets, which were to be used by NASA representatives to attempt to add or alter functionality specified in the POSIX 1003.4 draft in order to satisfy the requirements of the OSID.

3.0 Review of POSIX 1003.6

This effort involved a detailed analysis of the proposed POSIX 1003.6 working draft to determine if all functionality specified in the NASA-JSC Automated Information Security Plan (AIS) was satisfied. SwRI generated comments for any requirements which were not addressed in the POSIX 1003.6 draft. SwRI also generated comments on security requirements which were considered best addressed by manual procedures rather than with automated procedures.

OSID WORKSHEET

Control Number

Title

Optional features in POSIX 1003.4.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) Global
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

All of the major functions of POSIX 1003.4 are termed "optional". By definition, a strictly complying POSIX 1003.4 implementation would only be required to provide the functional interfaces (as opposed to any actual functionality).

Resolution

Add a new implementation definition which identifies a POSIX 1003.4 implementation which provides all listed functionality.

Rationale

Adding a new definition would reduce confusion for individuals trying to procure a system on the basis of POSIX 1003.4 compliance. The current definition may mislead individuals into procuring systems which do not adequately meet all requirements.

OSID WORKSHEET

Control Number

Title

Performance metrics.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) Global
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

The performance metrics need to be further defined for many of the 1003.4 functional areas. The definition should specifically state the metrics to be used and should provide algorithms for obtaining metric measurements.

Resolution

Complete the performance metric sections and introduce algorithms which explicitly show how the metrics may be measured.

Rationale

Most individuals involved in system procurement will be required to execute their own metrics on a system (as opposed to using measurements provided by the vendor). In the absence of a standard test suite, users will develop programs which measure the metrics. The amount of variability in these programs will be unacceptable unless the performance metrics are well defined and include basic algorithms. The advantage to algorithms (over code examples) is that they are language-independent. Actual code samples would be useful, but would be tied to a language (C or Ada).

OSID WORKSHEET

Control Number

Title

Pointers to static buffers.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 2.2.5
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

No POSIX 1003.4 function (or any POSIX function) should return a pointer to a statically allocated internal buffer.

Resolution

Add a paragraph which states that pointers updated and returned by POSIX functions will address dynamically allocated memory which may be directly used.

Rationale

A common problem in UNIX systems is that C function calls return pointers to memory statically allocated within the function. In order to safely use the data, the programmer must copy it to a new location before a subsequent call to the function. Designing functions which return pointers to dynamically allocated data allows more direct use (at the expense of forcing the programmer to free the memory).

OSID WORKSHEET

Control Number

Title

Changes to "General Terms".

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 2.3
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

There are several terms which must be modified or added in the "General Terms" section.

Resolution

Change or add the following terms:

- "binary semaphore" - (change) Change binary semaphore definition to describe the difference between binary and other (such as "counting") semaphores.
- "persistent" - (add) Persistence in this context refers to a special file (shared memory, semaphore) which remains available after the last close.

Rationale

These terms are required to improve understanding.

OSID WORKSHEET

Control Number

Title

Section 2 is not finished.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 2.2.4, 2.8, 2.9, 2.11
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

Many of the subsections in section 2 are incomplete. These sections are either blank or contain statements such as "The quick brown fox jumped over the lazy dogs".

Resolution

Complete the sections in question.

Rationale

These sections contain important information which must be completed for the final draft.

OSID WORKSHEET

Control Number

Title

Use of [ENOTSUP] and [ENOSYS] *errno* values.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 2.11.5
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

The description of the [ENOTSUP] ***errno*** value includes a description of the [ENOSYS] value. The description of [ENOSYS] is misplaced and is not correct.

Resolution

Describe the [ENOSYS] ***errno*** value in a separate bullet. Also, change the phrase "the implementation does not support all required functionality" to "the implementation does not support any required functionality".

Rationale

The description of the [ENOSYS] ***errno*** value is referenced inside the bullet for the [ENOTSUP] ***errno*** value. The description should be placed within its own bullet. In addition, the use of the word "all" is incorrect, as this error value indicates that the implementation does not provide "any" of the required functionality. This is opposed to the [ENOTSUP] ***errno*** value which indicates that only a particular function is not implemented.

OSID WORKSHEET

Control Number

Title

Binary semaphore persistence over a reboot.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[x] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 3.4.1.2, 5.4.1.2, 9.3.2.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The persistence of a binary semaphore over a reboot is implementation defined. This could result in non-portable code.

Resolution

Define persistence over a reboot in a standard manner. At worst, assume that the binary semaphore value is undefined.

Rationale

There is no benefit to allowing this behavior to be implementation defined. To be portable, an application must assume that a binary semaphore value is unusable after a reboot (worst case). To assume anything else would cause portability problems.

This comment applies to other IPC services which are implemented as special files and allow persistence (message queues and shared memory).

OSID WORKSHEET

Control Number

Title

Use of ***read()***, ***write()***, and ***lseek()*** on binary semaphores.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[x] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 3.4
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

Use of the ***read()***, ***write()***, and ***lseek()*** functions for a binary semaphore do not make sense.

Resolution

Describe such operations as undefined.

Rationale

Use of any of these functions does not make sense in the context for binary semaphores. The ***read()*** and ***write()*** function do not adequately relate to the wait (lock) and post (unlock) operations. The ***lseek()*** function has no use whatsoever.

OSID WORKSHEET

Control Number

Title

Inconsistent use of ANSI C.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 3.4.4.1, 10.3.1.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The function prototype for ***semifwait()*** does not use ANSI C format. This is not consistent with the majority of the document. A similar problem is found for the ***fcntl()*** function in section 10.3.1.1.

Resolution

Update the function prototype to use ANSI C format.

Rationale

ANSI C allows a function prototype to directly include the types of all parameters (as opposed to being typed on the following lines). This format is useful and is used throughout the document.

OSID WORKSHEET

Control Number

Title

Use of the ***unlink()*** function on a binary semaphore special file.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[x] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 3.4, 5.4, 9.3
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The effect of the ***unlink()*** function on a binary semaphore special file is not described in the document. What would occur if another process (or a command) was used to remove a binary semaphore special which was locked or unlocked? Would a process which was waiting on a binary semaphore receive the appropriate notification (via ***errno***) or would it suspend indefinitely?

Resolution

Define the effects of the ***unlink()*** function for the described instances.

Rationale

The affect of the ***unlink()*** function on persistent binary semaphores must be known to write portable code.

Similar problems apply to other IPC services which are implemented as special files and allow persistence (message queues and shared memory).

OSID WORKSHEET

Control Number

Title

Indication of unlock (post) of binary semaphore which is already unlocked (posted).

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 2.2.5
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The ***sempost()*** function does not indicate in any way in which the programmer can determine if the binary semaphore unlocked (posted) was already unlocked.

Resolution

Update the ***sempost()*** function to return an indication that the binary semaphore was already unlocked. This return should be transparent such that programs not interested in this case would not be affected.

Rationale

This ability is important in the 1003.4 implementation, as the "holder" of the binary semaphore (the locking process), will not be the only process which is allowed to unlocked. The described indication would allow a process to determine if another process inadvertently unlocked the binary semaphore, thus averting a potential deadlock or race condition.

OSID WORKSHEET

Control Number

Title

Typo in process memory locking section.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 4.3
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[x] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The process memory locking section, line 35 contains a typo. "Terminate a Process" is written twice.

Resolution

Correct the document.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Typo in shared memory section.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 5.3
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The shared memory section, line 28 contains a typo. The phrase "defined. binary" should be changed to "defined, binary".

Resolution

Correct the document.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Use of ***read()***, ***write()***, and ***lseek()*** functions for shared memory special files.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[x] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 5.4
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The behavior of the ***read()***, ***write()***, and ***lseek()*** functions should be defined. The document describes their behavior as implementation-defined.

Resolution

Define the behavior of these functions.

Rationale

These functions are reasonable for shared memory files and their behavior should be defined. Although shared memory will most commonly be mapped and accessed directly, it may be more convenient for certain programmers to use familiar ***read()***, ***write()***, and ***lseek()*** functions. If it is determined that this is unreasonable for general implementation, then the behavior should be undefined. At any rate, it is not reasonable to leave the behavior as implementation-defined.

OSID WORKSHEET

Control Number

Title

Implementation specific priority scheduling algorithm.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 6.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[x] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

Three scheduling types are required, but only two are defined. The third designated by SCHED_OTHER must be implemented to conform, but the algorithm used is implementation specific.

Resolution

Specify in OSID that SCHED_OTHER shall implement an "aging" scheduling algorithm.

Rationale

This implements the intent of 1003.4 which is to allow a separate non-real-time scheduling algorithm.

OSID WORKSHEET

Control Number

Title

Negative priorities may cause problems.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[x] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 6.3.1.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[x] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

Setpriority() and ***getpriority()*** returns the priority as an ***int***. This is a problem as certain implementations may allow negative priorities.

Resolution

Specify that priorities of -1 are not allowed.

Rationale

Negative priority levels are allowed as an implementation-defined feature. This is a problem as certain priority functions return -1 to indicate an error condition, making it impossible to differentiate a -1 error from a -1 priority.

OSID WORKSHEET

Control Number

Title

Function to return current event class mask.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 7.4.2.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[x] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

There is no straight-forward manner to retrieve the current event class mask. The ***evtprocmask()*** function provides this value, but via an odd combination of parameters.

Resolution

Add a new function which returns the current event class mask

Rationale

The manner in which the event class mask is retrieved via the ***evtprocmask()*** is a "back door" approach. A more simple and straight-forward function is required. A simple macro designed around the ***evtprocmask()*** would be suitable.

OSID WORKSHEET

Control Number

Title

Function types for ***evtsetjmp()*** and ***evtlongjmp()***.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 7.4.6.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[x] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The types shown for the ***evtsetjmp()*** and ***evtlongjmp()*** functions are incorrectly defined as ***int***.

Resolution

The return values for the ***evtsetjmp()*** and ***evtlongjmp()*** functions should be changed to ***void***.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

System timer setting.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[x] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.1.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The privilege to set a system timer is implementation-defined. Non-superuser's may or may not be allowed to set a system timer, as defined by the implementation. This leads to non-portable applications.

Resolution

Define the privilege required to set a system timer.

Rationale

A system timer is not normally available for user modification on multi-user systems. The most reasonable behavior is to limit system timer modification to the superuser. Alternatively, non-superusers may be allowed to set the timer (which would not make sense on a multi-user system). At any rate, the behavior needs to be consistent in order to allow development of portable code. The current behavior would force an application to assume that superuser status is required to set the timer.

OSID WORKSHEET

Control Number

Title

Timer section typos.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.1.2, 8.3.2.2, 8.3.4.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The listed section includes the type "***struct***itimercb" which should be "***struct*** itimercb".

Resolution

Correct the document. Note that section 8.3.4.2 includes two instances of the typo.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Timer resolution.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.1.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The document does not describe how a timer resolution is expressed. It is assumed that the resolution will be expressed in some increment of nanoseconds (1, 1000, 1000000, etc); however, this is not clearly stated.

Resolution

Clearly state the manner in which the timer resolution is retrieved.

Rationale

The resolution provided by the current implementation's timers is critical to all timer functions. The manner in which this information is retrieved must be clearly stated to allow other functions to be understood and effectively used.

OSID WORKSHEET

Control Number

Title

itimercbp / itimercb parameter typo.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.2.1 and 8.3.2.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The "Synopsis" section lists a structure instance named ***itimercbp***; the discussion in the following "Description" section uses a structure instance named ***itimercb***.

Resolution

Update the ***itimercbp*** value in the function prototype, as it is this value which appears to be incorrect.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Use of timer values which are not multiples of resolution.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.4.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The document does not describe what is done with timer increments which are not even multiples of the timer resolution.

Resolution

The exact treatment of such values should be defined. This may be via rounding up, truncation, or error return.

Rationale

The most appropriate behavior is difficult to select. Rounding up or truncation may cause problems for applications requiring critical timing. Returning an error is probably the best solution, but would require more work on the part of the programmer. The programmer would be required to retrieve the system resolution and only use valid multiples of this value.

OSID WORKSHEET

Control Number

Title

nanosleep() function.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 8.3.5.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[x] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

In many (most) implementations of the ***nanosleep()*** function, the timer resolution will not be able to sleep for small numbers of nanoseconds or accurately for odd multiples of the system timer resolution.

Resolution

Behavior of the ***nanosleep()*** function should be defined for such instances.

Rationale

The behavior should specify rounding up, truncation, or error return as consistent with the behavior of all system timers.

OSID WORKSHEET

Control Number

Title

Adding a function to purge all messages from a message queue.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 9.
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

One of the open issues is to add a "purge all messages from the queue". This would be a useful function.

Resolution

Define a function which allows a process to selectively purge messages in a queue.

Rationale

The direction of this open item is to add a function which returns the number of messages in the queue. Although useful, the programmer will still be required to purge the applicable messages from the queue. It would be more convenient to provide a set of functions which selectively purges messages based on ***pid***, ***type***, and other values. This function could purge all messages within a category (***pid***, ***type***, etc) if the corresponding value is a pre-defined constant (not 0 as this would be dangerous).

OSID WORKSHEET

Control Number

Title

Name of the ***mq_errno*** member.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 9.3.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The name of the ***mq_errno*** member of the message control block structure ***mqcb*** should be renamed to reflect its relationship to asynchronous event errors.

Resolution

Rename the ***mq_errno*** member to for example, ***mq_aserrno***.

Rationale

The existing ***mq_errno*** member name does not reflect that it pertains only to errors from asynchronous reads.

OSID WORKSHEET

Control Number

Title

Operation of ***mqsend()*** function with ***MQ_ASYNC*** flag.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 9.3.6.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[x] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

Using the ***MQ_ASYNC*** flag in an ***mqsend()*** function states that an asynchronous event notification will occur when the message has been received. It is assumed that event generation is automatic; however, this is not clearly stated in the document.

Resolution

Define the behavior more clearly.

Rationale

The behavior must be clearly stated in order to develop portable code. If the programmer assumes that the system will generate the message and it never occurs, then the message sender will never receive notification. On the other hand, if both the system and the receiver generate events, the sender will receive duplicate notifications.

OSID WORKSHEET

Control Number

Title

Performance metrics for Synchronized I/O.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 10.5
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[x] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The performance metrics Synchronized Input Time and Synchronized Output Time require the transfer of 1 megabyte of data. This creates a problem of manufacturers using very large caches to fool the metric. A better approach would be to specify a data size at least N*the largest cache in the system or specify that all caches and buffers shall be cleared before the performance metrics are executed.

Resolution

Add a statement to the OSID that all caches and buffers shall be cleared before performance metrics are executed.

Rationale

This will ensure performance metrics will be comparable between vendors.

OSID WORKSHEET

Control Number

Title

Clarification in *acancel()*.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 11.4.5.3
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[x] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

The meaning of a return value of 0 for ***acancel()*** is confusing. Line 259 states that a return value of 0 means the requested operations were canceled. This implies the event has occurred before the return from the function. Line 263 states that event notification is not given for asynchronous I/O cancellation.

Resolution

Rewrite line 263 to read "Returning from ***acancel()*** serves as event notification."

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Typo in introduction to Real-time Files.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 12.1
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[x] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

A typo appears on line 32 of paragraph 12.1. The line should read "with no changes necessary", not "with not changes necessary".

Resolution

Correct the document.

Rationale

N/A.

OSID WORKSHEET

Control Number

Title

Comments on Performance metrics in Real-Time files.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[x] OK
[ ] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) 12.5
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[x] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[ ] 12. Other

Description

In the Real-Time files section 12.5, the performance metrics Maximum Transfer Rate for Read Operations and Maximum Transfer Rate for Write Operations specifies the number of data bytes transferred by a percent of the available file system. This makes no mention of cache sizes and can give misleading results.

Resolution

Add a statement that all caches and buffers shall be cleared before performance metrics are executed.

Rationale

This will ensure performance metrics will be comparable between vendors.

OSID WORKSHEET

Control Number

Title

Structures described to be implementation defined.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[ ] Modification of Strictly Conforming POSIX
[x] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) B.1.2.2
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

Paragraph B.1.2.1 describes several structures as implementation-defined. This means that each implementation can define different structures required by threads. For portable applications, this is unacceptable.

Resolution

Add a statement which specifies that all structures in B.1.2.1 shall only be used in the default state.

Rationale

This will allow portability of code.

OSID WORKSHEET

Control Number

Title

General comments on threads.

Applicable Profile(s)

[ ] Real-Time (Onboard-DMS)
[x] Real-Time (Ground)
[ ] Other

Classification

[ ] OK
[x] Modification of Strictly Conforming POSIX
[ ] POSIX Extension Required
[ ] Non-POSIX Extension
[ ] Defn. of Implementation-Defined Behavior

POSIX References

[ ] 1003.1 Paragraphs(s)
[x] 1003.4 Paragraphs(s) B
[ ] 1003.6 Paragraphs(s)
[ ] 1003.8 Paragraphs(s)
[ ] Other Paragraphs

Subject(s)

[ ] 1. Process Env.
[ ] 2. Process
[ ] 3. Scheduling
[ ] 4. Events
[ ] 5. Timers
[ ] 6. Files/Dir.
[ ] 7. Input/Output
[ ] 8. IPC
[ ] 9. Networking
[ ] 10. System Adm.
[ ] 11. Security
[x] 12. Other

Description

The entire section on threads is still in flux. There are several open issues which will take time to resolve. Threads should be reviewed again at a later time.

Resolution

Track threads in the coming 1003.4 drafts.

Rationale

Threads are an important feature of 1003.4 and should be tracked to ensure their usability for ground support.

1.0 Introduction

This document interprets the requirements in Johnson Space Center's Automated Information Systems Security Plan (AIS) and determines if the requirements are met by the POSIX 1003.6 draft 3 extension for standard security (henceforth termed "POSIX 1003.6"). During the review of the AIS document, most of the detailed computer system-related requirements were found in Chapters 11 and 13. Some of the introductory chapters include high-level requirements which were later expanded in Chapter 11. For this reason, chapters 11 and 13 were exclusively used as the basis for the comments in this document. The basic organization of this document is to interpret each computer system-related requirement in the AIS and then describe whether or not the requirement is satisfied by POSIX 1003.6.

2.0 General Comments

From the review of the AIS, it appears that JSC will require the Discretionary Access Controls (DAC) and auditing. It does not appear that Mandatory Access Control (MAC) is required, as this is primarily required for systems which support different sensitivity levels of data. The AIS did not clearly indicate that this was necessary.

POSIX 1003.6 is not complete and does not yet address a number of important functional areas. While the Discretionary Access Controls (DAC) and auditing are fairly well developed, the Mandatory Access Controls (MAC) are incomplete and the Privileges section is effectively blank.

The auditing section of POSIX 1003.6 is fairly well-defined but has a limited scope. It does not address the manner in which auditing is enabled for users, files, or applications. It is not clear whether such functions should be part of POSIX 1003.6 or are more appropriately placed in the POSIX 1003.7 (System Administration) specification. POSIX 1003.6 depends a great deal on the POSIX 1003.7 (System Administration) specification. It is not clear whether administration of the security system will be part of POSIX 1003.6 or POSIX 1003.7.

POSIX 1003.6 does not address the role of the root (or superuser) user. The interaction between this privileged ID and the security system is undefined. It is not clear if there are multiple privilege levels (hierarchal system) or simply a single privileged ID which has access to the entire system (including security).

POSIX 1003.6 has significant effects on the POSIX 1003.4 (real-time) specification. In addition to the basic conflict between security and real-time response, it is unclear of the affects of DAC and MAC on the features provided by POSIX 1003.4, including message queues, shared memory, semaphores, timers, events, etc.

3.0 Detailed Comments

This chapter lists the section number and label for each of the relevant discussions in Chapters 11 and 13 of the AIS, provides an interpretation, and indicates whether or not the requirement is met by POSIX 1003.6.

11 Security Standards

11.3 Operating Systems

11.3.1 Operating System Controls

Interpretation

This capability will be provided via basic owner, group, and other permissions for read, write, and execute. Permissions will be attached to all objects (files and applications) for which protection is required. It will be the responsibility of users and administrators to assign and maintain the appropriate access controls.

Requirement

POSIX 1003.1 provides the owner, group, and other permissions for read, write, and execute. POSIX 1003.6 provides additional access control via Access Control Lists (ACLs), which are part of Discretionary Access Control (DAC). ACLs allow objects to include lists of user, group, and other permissions.

11.3.2 Clear System Programs

Interpretation

The AIS does not indicate when the "clear" operation must take place. It is assumed that this clear function will take place when the object is moved or removed from its current physical location. This requirement is applicable to all forms of random access storage, including disks (for when an object is removed/moved) and primary memory (when an application explicitly or implicitly [via termination] frees memory space).

This is a reasonable concern, as in most implementations, it is possible to allocate disk or memory space without actually clearing or writing any data. In this instance, the old sensitive data could appear in an another object (a file or memory buffer). An application could repeatedly allocate disk or memory space in an attempt to locate sensitive data.

This requirement will most appropriately be provided via a remove function which will clear (zero out) space occupied by sensitive data. Note that it requires an order of magnitude more time to clear data space.

Requirement

POSIX 1003.6 references the term "object reuse", which indicates that objects (files, disk space, etc.), are reusable if they do not include any residual data. The actual mechanism for this was not found in the specification. This should be a feature of MAC, which is used on systems with multiple sensitivity levels of data.

11.3.3 Shutdown and Restart

Interpretation

It is not clear what this requirement entails. Systems which have failed or are down for normal maintenance tend to be very vulnerable (at least UNIX systems). Unless special steps are taken, it is often possible to boot from an alternate location (such as a tape, secondary disk, or floppy). This is an obvious problem as the software on the alternate media would not be constrained by the security system. Another security problem is that many UNIX systems allow single-user mode to be initiated (booted) without a specification of a password.

Requirement

POSIX 1003.6 does not specify the security provided outside of the normal POSIX operating environment. Security outside of the operating system is inherently implementation dependent and would be difficult to specify the interfaces in a standard. This is however a valid requirement and should be addressed in either POSIX 1003.6 or 1003.7.

11.3.4 Recovery Management

Interpretation

Functions which need to be recovered after a system has failed should be defined when the system is designed so that any redundant information which is necessary for recovery and historical log keeping facilities can be built into the system.

Requirement

POSIX 1003.6 does not provide an interface for recovery management. This subject may be more appropriately covered in System Administration, 1003.7.

11.4 JSC Standard On Controlled Access Protection

Interpretation

Systems containing sensitivity level 2 or 3 objects will provide individual authentication and accountability by use of unique user IDs and passwords. The system will prevent unauthorized access or modification of this information.

Audit trails will be generated for all activity on privileged IDs and for user-selectable periods as required for suspected violations and spot checks. It must be possible to enable auditing for a given user or object.

User access to objects will be provided by permissions and ACLs. Maintenance of permissions is the responsibility of the user and system administrator.

Requirement

POSIX 1003.6 includes the concept of sensitivity levels via the Mandatory Access Controls (MAC). MAC provides "labels" which may be attached to objects and subjects (processes). The labels define different sensitivity levels and provide rules for access control. Use of MAC will be necessary if a system uses different sensitivity levels. It is not clear from the AIS whether or not a given system includes different sensitivity levels.

POSIX 1003.6 provides auditing functionality. The specification primarily defines an example interface which allows an application or function to generate audit records in a standard format. The specification does not describe how auditing is enabled for a user, file, or process.

POSIX 1003.6 provides access control via basic permissions and ACLs.

11.6 ID Administration And Ownership

11.6.1 Issuance

Interpretation

JSC procedure. For discussion of the different types of ID's, refer to section 11.6.4, which deals with the responsibilities of the different types of users.

Requirement

POSIX 1003.6 does not directly support the 5 types of user IDs. However, several of the types may be logically implemented with permissions and ACLs.

11.6.2 Annual Revalidation

Interpretation

JSC procedure. This procedure would be aided by automated support which retains the dates at which time each user ID was last utilized. This is a simple accounting procedure and does not add an appreciable amount of system overhead. This information would be useful in determining which accounts could be removed during the annual (or periodic) revalidation procedure.

Requirement

POSIX 1003.6 does not specify saving of any user-specific time stamp information to the file containing user IDs. If such automated record keeping is required, then it must be determined whether it belongs in POSIX 1003.6 or POSIX 1003.7.

11.6.3 Expiration

Interpretation

JSC procedure. This procedure will also benefit from automated accounting. Each user ID and password could have an associated creation time and an expiration period. When this period expires, a notification could take place or the system could automatically disallow subsequent access attempts. This mechanism would also allow the expiration to be set for different types of users (short expiration for privileged and high-sensitivity users and long for others).

Requirement

POSIX 1003.6 does not specify use of expiration times for user IDs and passwords. Again, if this function is required, it may be added to POSIX 1003.6 and POSIX 1003.7.

11.6.4 ID Owner Responsibilities

11.6.4.1 Personal ID Owner Responsibilities

Interpretation

Leaving a terminal session unattended is a problem and must be controlled with procedures. An electronic alternative is to require a "watchdog" daemon which logs the user off after a defined period of inactivity. Such daemons add overhead and often are forced to take unpleasant actions against users (such as terminating an edit session or a CPU bound process).

Protecting user data, keeping passwords secret, reporting expired IDs, and reporting disclosed passwords are controlled via procedures.

Requirement

POSIX 1003.6 provides permissions and ACLs for protecting of user data. It is however up to the discretion of the user and system administrator to use these mechanisms to protect data.

1.6.4.2 Privileged ID Responsibilities

Interpretation

The AIS implies that multiple privileged user IDs are required. This is necessary to allow individual accountability, as if all privileges were through a "root" ID, it would not be possible to track the individual responsible for a security violation.

All actions taken with a privileged account should be audited automatically by the system. The audit process must be real-time and be assured to complete before the privileged action is allowed to take place. A reasonable procedure will be to utilize a specially protected write-only device (printer or disk) to which audit information is output.

Privileged ID passwords should expire more frequently than normal user IDs. It may also be necessary to review IDs more often than annually.

Providing "templates" of privileges for different tasks would be useful, but difficult to implement. It may be more appropriate to designate such users as non-privileged and assign ownership of the appropriate files to allow access.

Privilege granting is a JSC procedure.

If additional environmental controls are required, it would be possible to implement a system which only allows privileged access during periods of time or only when a designated system administrator is currently logged on to the system.

Requirement

POSIX 1003.6 does not yet define privileged access. Although the scheme for privileged access is not known, it is suspected to be similar to the UNIX usage of the "root" ID. This is a problem as there is no way to account for individual actions, as all privileged users would use the same ID. A solution is to require an interface in 1003.6 which specifies creation of individual IDs, each with the equivalent of root access. Another solution is to disallow direct root access. In order to use root, a user must first log in as a normal user and then change to root.

POSIX 1003.6 does not specify the relationship between privileged use and protection of security related files. In particular, will it be possible for a privileged user to circumvent the security system (by updating or disabling the audit system)? This is probably possible, but should be noticed with real-time audit record generation.

11.6.4.3 Project ID Responsibilities

Interpretation

It appears that a project ID is the same as a personal ID, except that the owner is responsible for insuring that the project data is not accessed outside of the project group. This may be performed by setting the permissions and ACLs on data objects so that project users alone can access the data as required (reading, writing, and executing). The project ID user could also allow new users to access the project data by updating the group or changing ACLs.

Reviewing "all access lists" most likely refers to determining if all users still need access to data (still belong in the group) and if all permissions are correct.

Requirement

POSIX 1003.6 provides all functionality necessary to support logical project IDs. A project would most likely correspond to a POSIX 1003.1 group.

11.6.1.4 System Service IDs

The distinction between system service ID and a privileged ID is not clearly defined. Does a system service user require privileged access? A user responsible for maintenance of system software will normally require privileged access, especially if the intended definition of "system" includes operating system, security, graphics, or other software.

It appears that the system service ID may be interpreted as a special form of privileged user which is temporarily provided to allow vendors and support personnel to periodically update the system software. The statement "the ID Manager may grant authorization to login to the ID to other users" implies that this ID will be temporarily provided to users who must perform system maintenance. This of course could cause problems if the password is not immediately modified after the ID is used.

All activity on a system service ID should be audited as described for a privileged user ID. Passwords should be treated in the same manner as privileged IDs, rather than in the same manner as personal IDs.

An alternative approach is to set the permissions of all files pertaining to a "system" to be owned by the system service ID user. In this way, that user could change the software as needed. Again, this is not foolproof, as unless all objects are isolated, it would be a problem to add new objects or directories.

Requirement

POSIX 1003.6 does not directly specify this type of user ID. As indicated, it may be implemented via a privileged ID or a user ID which owns all relevant objects. It may be most appropriate to remove this ID type and classify it as privileged.

11.6.4.5 Generic ID Responsibilities

Interpretation

The distinction between generic and personal IDs is not clear. It does not appear that generic IDs are "guest" accounts, as this is a basic violation of a secure system. Generic IDs also do not appear to be common IDs which are shared by multiple users, as this would prevent individual accountability. Rather, generic IDs are similar to personal IDs, except that a particular users access is fairly short-lived (such as a training exercise in which a temporary ID is assigned for use).

It is a necessary procedure to immediately change the ID's password once a user has completed its use. The expiration time on such ID passwords should be very short.

Requirement

POSIX 1003.6 provides the functionality described (assuming the interpretation described).

11.6.5 ID Management

Interpretation

JSC procedures.

Requirement

N/A.

11.6.6 Non-trivial Passwords

Interpretation

Most of the constraints described are reasonable and should be implemented as JSC procedures. This is primarily due to the fact that whether or not a password is "trivial", is a subjective decision which cannot be automated.

An additional constraint is a procedure which insures that users do not use the same password for multiple systems. This is a serious problem when users are active on systems at different sensitivity levels. On non-secure development systems, users often allow their passwords to be known

by other users. If the same password is used on a secure system, it would be possible for an unauthorized user to gain access.

When a new system is received, it is a sound procedure to insure that all IDs (especially privileged IDs) are updated to use valid non-trivial passwords. When shipped, most systems will only include a few IDs necessary for system operation. This includes a privileged ID which will not be password protected (UNIX root).

Requirement

POSIX 1003.6 does not describe use of passwords. This is also absent from POSIX 1003.1. If any of the constraints described are to be automated, modification must be made to the appropriate specification. It would be reasonable for the system to implement the following constraints:

- Password not equal to user ID.
- Password does not consist of obvious strings (the strings are listed in a file which may be updated by system administrator).
- Updated passwords must not be successive - during password reviews, the system must insure that changed passwords are actually different. It is also necessary to prevent a user from changing to a temporary password and then back to the previously used password.

Access and maintenance of passwords may be most appropriately described in POSIX 1003.7

11.9 System Security

User Identification

Interpretation

JSC procedure.

Requirement

N/A.

Application Independence

Interpretation

This capability will be provided via basic owner, group, and other permissions for read, write, and execute. ACLs may be used if necessary.

Requirement

POSIX 1003.1 provides the owner, group, and other permissions for read, write, and execute. POSIX 1003.6 provides additional access control via ACLs.

Maintenance of the Security System

Interpretation

The definition of "real-time" is unclear? Is this true real-time response or is it simply "on-line" in which the security record updates are not queued? Queueing records may allow an individual to intercept and remove them before they are logged. If the covert action is to disable the system, then if the record is not logged first, the system will fail before the violation is recorded. This case demonstrates that true real-time logging is required (at least in special cases).

Access controls include basic permissions and access control lists. Operator capabilities will be limited via these access controls. Password changes will be audited and only allowed for privileged users (and possibly for a user's own ID).

It is assumed that "system programmers" are privileged users. There must be a mechanism which prevents privileged users from affecting audit files (at least without the knowledge of the system).

Requirement

POSIX 1003.6 does not specify whether or not audit record updates are logged in "real-time". This is a reasonable requirement.

POSIX 1003.6 defines the basic access controls and ACLs. The specification does not describe any means of controlling access to passwords.

POSIX 1003.6 does not specify how audit files are protected from privileged users.

Security System Administration and Monitoring

Interpretation

JSC procedures. "Self-auditing" is assumed to involve a privileged user having access or control of audit information generated for himself. Self-auditing could be prevented by disallowing access to security files, real-time audit record logging, or logging of all privileged actions to a write-only device.

Requirement

POSIX 1003.6 does not specify the manner in which self-auditing is prevented.

Auditability

Interpretation

Security auditing must be provided and it must be possible to enable/disable auditing for the following:

- User ID (including privileged IDs)
- Process
- File (such as the password file)

The system must allow audit for a controlled period of time. The system must also provide a set of functions which allow the audit records to be reviewed in a straight-forward manner.

Requirement

POSIX 1003.6 defines the basic functions required to generate audit events. These functions may be included in a trusted application to generate audit records. POSIX 1003.6 does not specify the system utilities which allow control of the audit system for user IDs and objects. This is apparently to be covered in the POSIX 1003.7 specification or a later draft of POSIX 1003.6.

POSIX 1003.6 does not specify any functions which allow review of the audit records. The specification defines a standard format which will allow other functions to review the audit data. Such review functions/commands will probably be found in POSIX 1003.7.

Terminal Operator Training

JSC procedure.

Requirement

N/A.

11.9.6 Password Usage

11.9.6.1 Composition

Interpretation

An automated password system is required to verify that the generated or selected password is composed of characters from a subset of at least 10 characters taken from the set of 95 graphics characters. It is unclear whether this automated password system is generating passwords for users and how the subset of 10 characters is selected.

Requirement

POSIX 1003.6 does not provide this function.

11.9.6.2 Length Range

Interpretation

This section calls for the password to be at least six characters long and to allow for a minimum of 10,000 possible passwords when used in conjunction with the number of characters in the composition subset. An automated password system shall verify that only passwords having a length within the acceptable length range shall be generated or selected whenever a password is created or changed. The selected password length will be an indication of the level of privilege associated with the password.

Requirement

POSIX 1003.6 does not provide this function.

11.9.6.3 Lifetime

Interpretation

Passwords shall have a maximum lifetime of 6 months.

Passwords which are suspected of being compromised should be replaced within one working day from the time of suspicion. Passwords should be deleted or replaced within three working days from the time that an owner is no longer an authorized system user, or any one of a set of owners is no longer authorized access to the data.

Passwords forgotten by their owner shall be replaced, not reissued. This is a procedure.

An automated password system shall allow the ID Administrator to delete or replace a password and will be capable of maintaining a record of the password transaction.

Requirement

There is no reference in the POSIX 1003.1 or the 1003.6 document to a function which will allow update of a password. There are functions for retrieving, but not for setting.

The only reasonable approach to monitoring the life of a password is to store the date of the last change, calculate an expiration date based on the time of creation or change, and then store the expiration date also. An automated routine could then be used to periodically check for passwords which have expired. The storage of these dates could be useful for other security requirements as well.

11.9.6.4 Source

Interpretation

All passwords must be randomly selected from a list of acceptable passwords or a string which is not at all related to a user's personal history. This is a JSC procedure.

Passwords selected or created shall be tested by the automated password system to assure that they meet the specifications of composition and length established for the AIS system before they are accepted as valid passwords.

Requirement

POSIX 1003.6 does not provide these functions.

11.9.6.7 Ownership

Interpretation

JSC procedures.

Requirement

N/A.

11.9.6.6 Distribution

Interpretation

Passwords are created and assigned to users by the automated password system. Any time a change is made to the password, an audit record containing the date and time of the password change and the identifier associated with the password is created and made available to authorized security personnel.

When the passwords are distributed from the password source, the temporary storage of the password should be erased, and the permanent storage of the password should only be accessible to the owner and the protected password system. There is no mention of privileged users being able to access the passwords.

Requirement

POSIX 1003.1 has deleted the storage of encoded passwords from both the *passwd* and *group databases*. The standard provides functions for keyed lookup of passwords so that it may not be possible for an application to browse the system databases indiscriminately. Where the passwords are stored is not clear.

11.9.6.7 Storage

Interpretation

Stored passwords will be protected in such a way that only the password system is authorized access to a password. Passwords that are encrypted before they are stored shall be protected from direct substitution.

Requirement

POSIX 1003.6 does not provide this function. Passwords are encrypted before they are stored and are never decrypted. Whenever an entered password needs to be verified, the entered password is encrypted and then compared to the stored encryption. This helps to protect the access to the password.

11.9.6.8 Entry

Interpretation

The system will not echo a password when being entered by the user. It must not be possible for the user (or another user) to modify this behavior.

It must not be possible to emulate the login process on a terminal in order to "steal" passwords from a user attempting system access.

The number of allowed password entry attempts (retries after incorrect password entry) shall be limited to a number selected by the system administrator. An ID shall be locked out in response to exceeding the maximum number of retires specified by the system administrator.

Invalid logins to a user ID shall be tracked and reported by the security system. Notification of invalid access attempts should be made to the security administrator and the user.

Requirement

POSIX 1003.6 does not provide any of the listed functions.

11.9.6.9 Transmission

Interpretation

It is not clear what this requirement is trying to address. The entered password will be encrypted (if necessary) and then compared to the stored password. All movement and comparisons will occur in the address space of a process. It is not likely that this data space will be violated.

Passwords transmitted between the place of entry and the place of comparison with the stored password shall be encrypted at the place of entry if the data that the password is protecting is encrypted at the place of data entry.

Requirement

Since it is not clear what this requirement is addressing, it is unknown what the requirement in POSIX may be.

11.9.6.10 Authentication Period

Interpretation

Personal passwords shall be authenticated each time a claim of identity is made, e.g., when "logging in to" an interactive system.

Access passwords shall be authenticated during the initial request for access to protected data.

Requirement

There is no mention of password authentication in the POSIX 1003.6, however most UNIX systems will request a password whenever access is attempted.

13. AIS Security Infractions Violations

13.2 Detection

13.2.1 System Resource Monitoring

Interpretation

The basic permissions and ACLs may be used to control access to most system resources (such as files, devices, and system applications). Abuse will be prevented by not allowing access

to these resources. Abuse of resources such as CPUs, printers, and storage devices is prevented by use of quotas which limit the amount of access, time, or space allocated to a given user.

Requirement

POSIX 1003.6 does not include any mechanism for controlling resource allocation. Many UNIX systems provide an accounting/quota mechanism which may be used for this purpose. This however is more appropriately placed in POSIX 1003.7.

13.2.2 Audit Logs

Interpretation

All actions taken by privileged users should be audited. While this may not be the constant operating state, it must be possible to enable or disable this operation.

Requirement

POSIX 1003.6 provides the basis for auditing. As previously described, the POSIX 1003.6 specification does not address the manner in which auditing is enabled for a given user or object (file or application). This however is a valid requirement and must be present in the POSIX 1003.6 or 1003.7 specifications.

As previously described, it is not clear how POSIX will prevent privileged users from accessing audit logs.

13.2.3 Exception Reports

Interpretation

Generation of exception reports is useful for picking out the audit events which pertain to a suspected security violation.

Requirement

POSIX 1003.6 does not specify any functions or commands which may be used for audit log post-processing. POSIX 1003.6 does however specify a standard format for audit records which will simplify development of functions and commands which provide post-processing. Such functions will most likely be described in POSIX 1003.7.

13.2.4 Operator Logs

Interpretation

Operator logs would be a useful way of monitoring system activity. Real-time generation and review of audit records will prevent even a privileged user from adversely affecting the security system. In the absence of a human operator, the audit records could be output to a printer or other write-only device.

Requirement

POSIX 1003.6 does not specify the manner in which audit records are generated. Whether or not the records are generated in real-time and the ability to route to a specific destination appear to be implementation-defined. Generation of real-time audit records is a valid requirement.

1.0 Introduction

A current problem faced by NASA and other organizations is a large amount of code developed for systems that will not be supported in the future. In NASA's case this is FORTRAN code not compatible with the workstation/UNIX environment. One way to still use the FORTRAN code is to isolate nonstandard FORTRAN functionality into C subroutines. This does not provide a portable solution but isolates the problem to interlanguage support. The only way to ensure that interlanguage support is vendor independent is through an independent standards organization. In this case POSIX is the committee most likely to be developing interlanguage standards.

2.0 POSIX

2.1 FORTRAN Bindings

John McGrory, chairman of the FORTRAN Bindings committee, was contacted about interlanguage support and indicated that his group is not pursuing interlanguage support. Mr. McGrory is willing to help introduce any efforts of introducing an interlanguage specification. He referenced me to Rick Anderson of Boeing Computer Services. Mr. Anderson has completed a first draft of a proposal to allow FORTRAN routines to call C subroutines. The proposal is currently being reviewed at Boeing and he is sending a copy of the proposal to SwRI. He was very interested in having other groups support the proposal.

The FORTRAN bindings specification will be voted upon in August of 1990. It is expected to require a second vote in December of 1990. This is for bindings to 1003.1. Then the committee will begin work on bindings for any POSIX standards that are completed. This is expected to be the real-time specification (1003.4) and the revision of the 1003.1 specification.. The completion of this work is expected in second quarter of 1991. Bindings for the most recent version of FORTRAN (8X or 90, depending on who you talk to) is expected to begin at the end of 1990.

2.2 Ada Bindings

Steven Deller is the current chairman for the committee on Ada bindings. Terrenace Fong is the previous chairman and the current vice-chairman. Mr. Fong indicates that interlanguage support is not being addressed by the bindings committee and there is not a group preparing a proposal. The bindings specification will be voted upon in the summer of 1990. After the standard has been adopted, the group will start work on the bindings for other POSIX standards that have been completed. Mr. Fong also commented that the committees will begin a language independent version of the POSIX standard when the current work has been completed.

3.0 Conclusions

At this time the interlanguage issue is a long way from being solved. Beoing may make a proposal, but that will take a long time to finalize and be supported by the industry. In a shorter time frame, the FORTRAN bindings specification will be completed. All functionality of 1003.1 will be available from FORTRAN compilers. At this time, the existing FORTRAN code can be rewritten to the POSIX standard. This will give access to system routines directly from FORTRAN and be portable to other POSIX compliant systems.

4.0 Contacts

Below is a list of contacts used in researching the interlanguage support issues:

Lisa Granoien
IEEE Computer Society
(202) 371-0101

John McGrory
POSIX FORTRAN Bindings Chairman
(408) 447-0265

Rick Anderson
Boeing Computer Services
(206) 865-3523

Terrance Fong
POSIX Ada Bindings Vice-Chairman
(602) 533-2873

1.0 Introduction

SwRI has developed a load sharing prototype as proof-of-concept for a capability identified in the Concept Executive specification. Load sharing refers to the ability to view a collection of workstations as one loosely coupled multiprocessor. Load sharing allows jobs to be scheduled on those workstations which have the highest amount of available resources.The load sharing prototype developed by SwRI supports the following capabilities:

- Efficient and more balanced utilization of all workstation processors and associated computing resources.
- Utilization of idle or underutilized workstation's computing resources.
- Utilization of dedicated compute servers for processing of user jobs.
- Allows users on older, less powerful workstations to take advantage of capabilities on newer, more powerful workstations.
- Allows user level control of a local workstation's participation in the load sharing activity.

This design is for a load sharing prototype, as opposed to a load balancing prototype. Load sharing means that underloaded workstations will give up some of their resources to support other, overloaded workstations. No attempt will be made to evenly balance the load across all workstations.

2.0 Functional Design Overview

The functionality of the load sharing prototype is divided into the following three classifications:

- Status collection - the process of retrieving status information for the current and remote workstations.
- Scheduling - the process of using status data to schedule user jobs locally and on remote workstations.
- User interface - the process which allows a user to submit, control, and monitor scheduled jobs.

The load sharing design is fully decentralized. All status collection and scheduling is performed in a distributed manner on each participating workstation. This is in contrast to a centralized status collector and scheduler. The decentralized approach was selected for the following reasons:

- A centralized design is much less fault-tolerant. In this design, if the scheduler fails, the entire system fails.
- Use of a centralized design delays process execution, as each job request must be transferred to the central scheduler.
- There is no dedicated host in the target network which would be a suitable central scheduler.
- Although a central scheduler can make better choices about the optimum target workstation, a decentralized approach is more than adequate.

The load sharing design supports a heterogeneous environment. This means different workstations from the same vendor and different vendors within the same environment are supported.

2.1 Status Collection

The load sharing prototype utilizes an approach in which status information is periodically "advertised" to all workstations in the local network. This approach is in contrast to status querying, in which when a job is ready to run, the source workstation queries one to many other workstations in an attempt to find a suitable (or the best) location for scheduling. The advertising approach was selected due to the following advantages:

- Availability of network broadcast messages which allow one datagram to be sent to all workstations in a local network.
- Provides the most efficient approach in local network containing a relatively small number of workstations.
- Provides much better response, since all status information is already available for job scheduling.
- Very efficient if broadcasts are only made when a workstation's status changes significantly.
- The advertising approach is more scalable (assuming that status data is not indiscriminately broadcast).

- Querying is inefficient, as an overloaded workstation is forced to become more overloaded in order to query destination workstations.
- Although querying can provide more up to date information, advertising is more than adequate.

The "load" calculated for a workstation is an aggregate value of the available processing, memory, and other pertinent resource information. Each workstation includes parameters which define its processing capability. The parameters include:

- Total processing capability (MIPS, CPU's, cache, etc.).
- Total memory.

2.2 Scheduling

The high/low algorithm is used to determine when a workstation is able to accept remote jobs and when a workstation will submit jobs for remote execution. The high/low algorithm involves maintaining a high and low load mark on each workstation. When a job is scheduled, it will be run locally unless the local load is greater than the high mark. Conversely, a workstation will only allow introduction of jobs if its load is less than the low mark.

The high and low marks are adjustable by the user. This allows users to control the degree of participation in load sharing.

Adjusting the high and low marks allows a workstation to be completely independent of the load sharing system. Adjustment also allows effective use of compute servers.

The high/low algorithm allows scheduling of jobs on workstations which are not completely idle. Any workstation whose load is less than the low mark is considered available to support remote jobs.

2.3 User Interface

The user is provided with a Motif-based user interface which presents the following pertinent information:

- All active hosts and their respective load and parameters.
- Local parameters (such as the high and low mark).
- List of all active jobs (local and remote).
- List of all status messages.

The user interface also allows control of all jobs. This includes the ability to perform each of the following functions:

- Scheduling of jobs.
- Termination of jobs.
- Changing the priority of jobs.
- Updating the local high and low marks.

3.0 Implementation Overview

The load sharing prototype was developed on Sun 4's and Masscomp 6000 series workstations. The complete list of hardware is:

- (1) Masscomp 6650 running RTU 4.0
- (1) Masscomp 6350 running RTU 4.1
- (4) Sun 4/60's running SunOS 4.0.3
- (3) Sun 4/65's running SunOS 4.1
- (1) Sun 4/330 running SunOS 4.0.3
- (1) Sun 4/150 running SunOS 4.0.3

Note that a bug was discovered on the Masscomp's which prevents use of a specific message queue feature. A temporary work around for this problem was implemented on the Masscomp. The implementation of the load sharing prototype is characterized by the following:

- The existing NASA prototype for Health and Status collection (has) was used to collect raw status information.
- TCP/IP-based network protocols are being used for network connectivity. This is necessary to support basic communications (sockets, broadcast, etc.), NFS, and X Windows.
- X Windows is used as the basis for job to source connectivity. Remote jobs not needing X are shelled with a terminal emulator (xterm). The user interface uses the Motif widget set and depends on the Motif window manager (mwm).
- The Network File System (NFS) is used to establish a consistent file system across all operating workstations.
- During normal operations, in which no scheduling activity takes place, less than 1% of available processing capability is used.
- Network interfaces are fully isolated from application code. This allows ISO to be easily introduced at a later time.
- All network data is passed in the form of UDP datagrams. No point-to-point connections are maintained.

The main portion of load sharing information is held in shared memory. All local message passing is performed via a single message queue. The shared memory segment used is relatively small (approximately 20K).

The load sharing prototype consists of five daemon processes which are always active and one user-initiated process. These processes and their responsibilities are described in the following sections.

3.1 Initialization (ls_init)

This daemon process is responsible for initialization of the load sharing system. This process performs the following:

- Initializes all resources (network, shared memory, semaphore, and message queue).
- Executes all other daemon processes.
- Waits for termination signals from children. Upon termination, the appropriate child is respawned.
- Upon receipt of a SIGHUP signal, ls_init performs an orderly termination of child processes and removal of all resources.

3.2 Health and Status (has)

This daemon process is responsible for retrieval of raw status information. The has process used for load sharing is a variant of the prototype developed by Ford Aerospace. The has process was modified to support both Suns and Masscomps.

3.3 Status Advertisement (ls_status)

This daemon process is responsible for advertising status information for the current workstation to other workstations in the network. This process:

- At a defined interval, retrieves raw status information saved in shared memory by the has process.
- At another defined interval, all accumulated status information is averaged into an aggregate load value. If this load value is "different" from the last advertised load value, the new information is broadcast on the network.
- If no information has been broadcast for a defined "heartbeat" interval, information is broadcast to allow other workstations to know that the current workstation is still active.
- Checks if any remote status information has arrived. If so, update local shared memory.
- At another defined interval, check the local list of hosts to determine if any are "stale" (no heartbeat received).

3.4 Network Interface (ls_net)

This daemon process retrieves all load sharing datagrams from the network. These datagrams are placed into message queues which allow the requests to buffer properly. This function is isolated in a separate process to insure that no datagrams are lost.

3.5 Job Scheduler (ls_scheduler)

This daemon process accepts job control requests and either executes the requests locally or routes them to the appropriate workstation (based on load). This process waits on a message queue for arrival of requests and responses. Requests originate from both local or remote users. The requests processed include:

- Schedule request - runs the job locally or routes it to the appropriate remote workstation.
- Termination request - if local, terminates the job; if remote, routes the request to the appropriate remote scheduler.
- Set priority request - if local, updates the priority; if remote, routes the request to the appropriate remote scheduler.

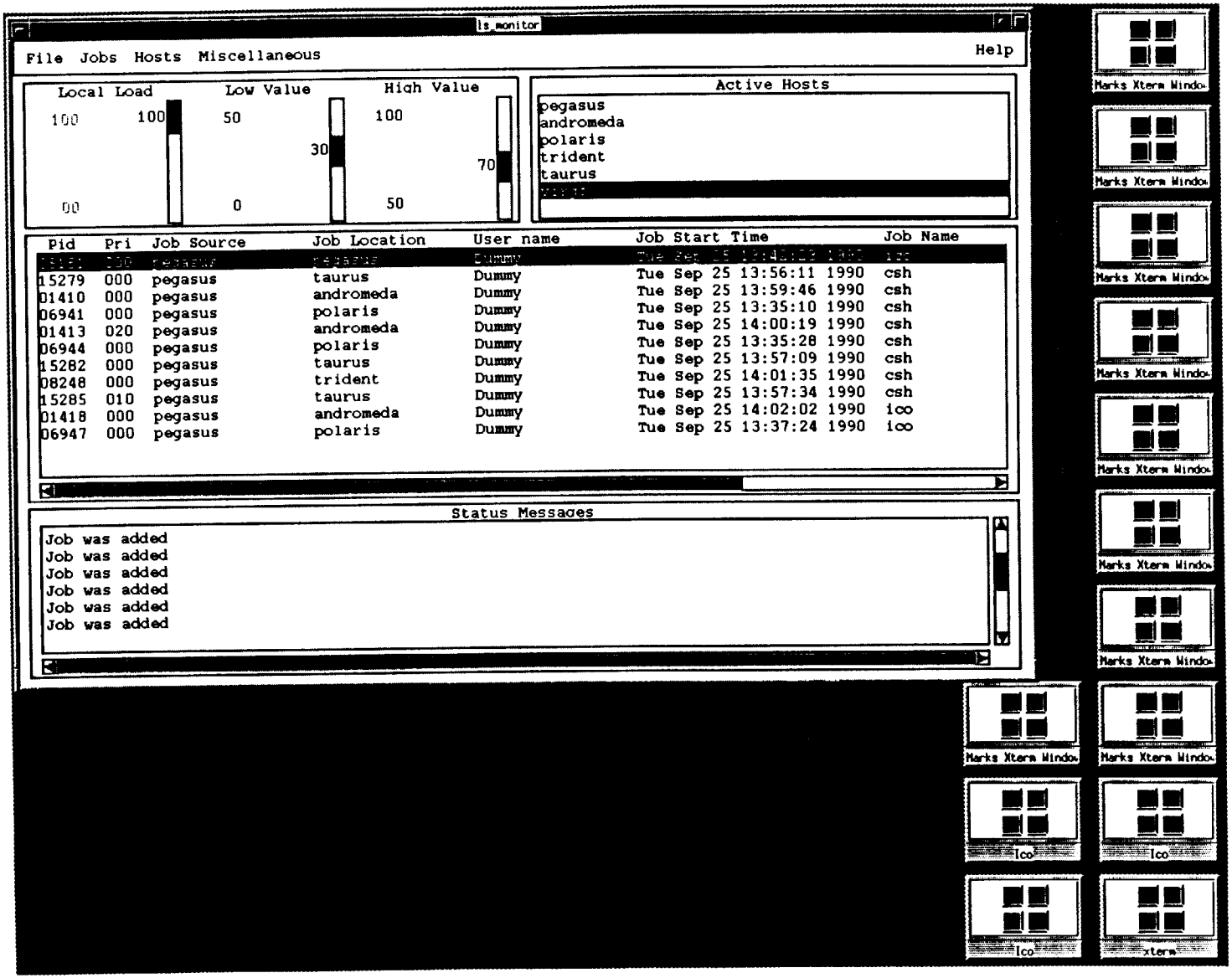
Changing the state of a remote job causes generation of a response which is returned to the local user. This allows the local user to be informed of the change in job status.

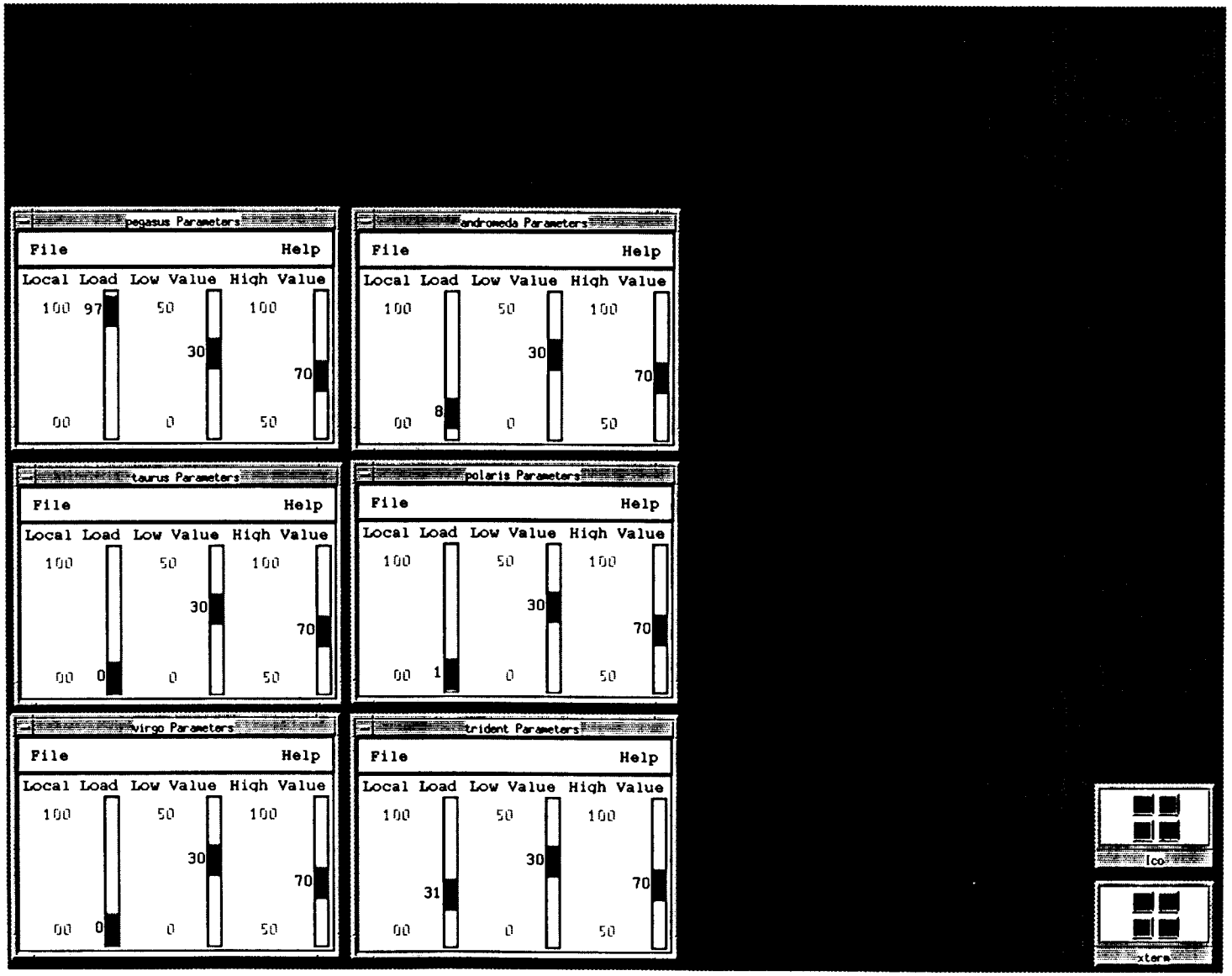
An occasional problem with the job scheduler occurs when a job is submitted to a remote workstation on which the load suddenly crosses the low mark. In this case, the job must be transferred to another workstation. This "bouncing" of a job will occur infrequently and is a natural effect of the status advertisement approach, in which local status information is stale for a brief period. To prevent a job from bouncing indefinitely, each job request includes a value which indicates the number of bounces. When this value reaches the maximum, the job is forced to execute on the original workstation.

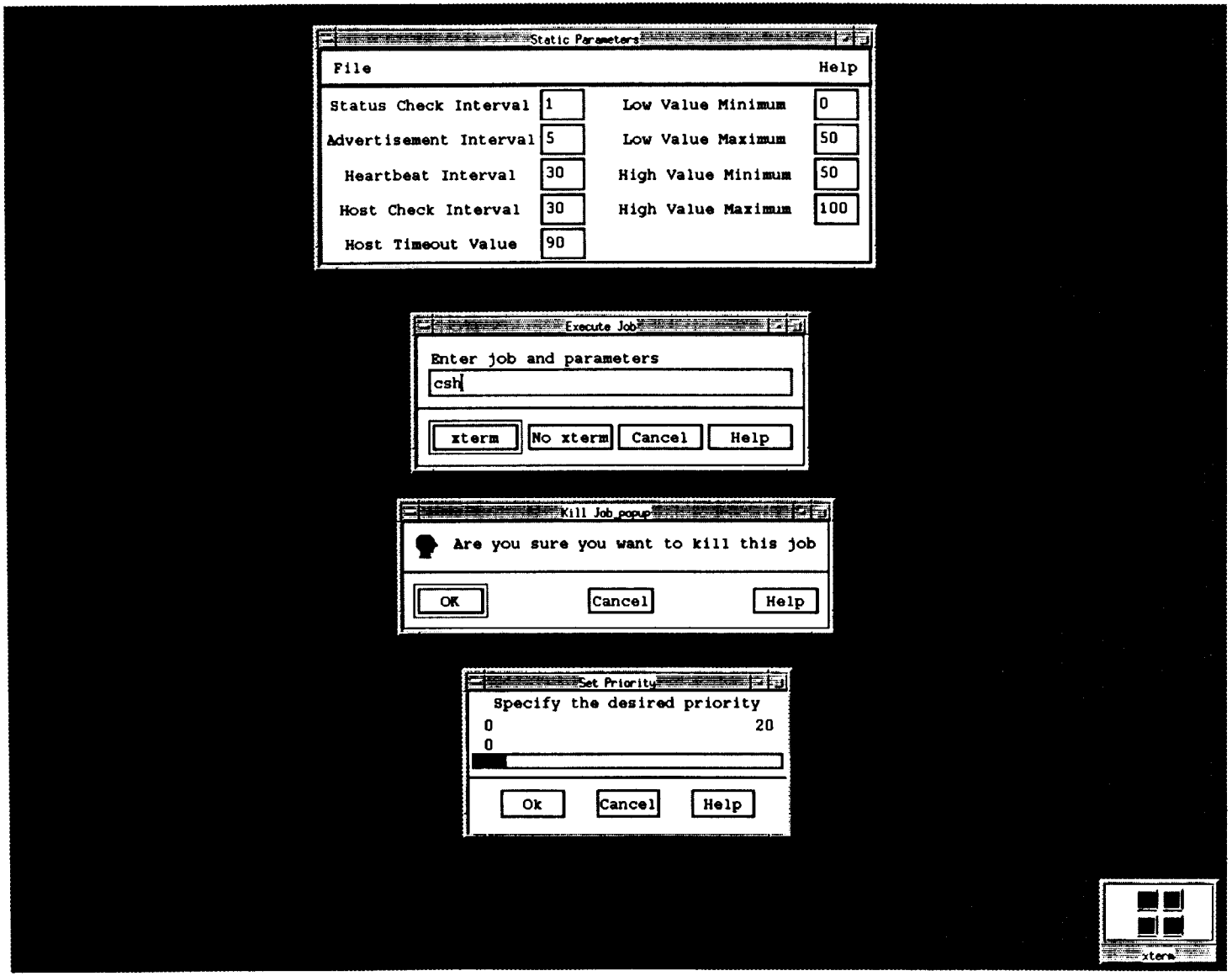
3.6 User Interface (ls_monitor)

This user-initiated process allows the user to use and monitor the load sharing system. This Motif-based process presents all pertinent information and allows jobs to be scheduled, monitored, killed, and have priorities updated. The following three pages provide sample displays which illustrate:

- The main display with several jobs running locally and on remote workstations. The active jobs are represented by icons on the right side of the display.
- Several host popup windows which display the high/low marks and load of the workstations participating in load sharing.
- The static parameters, job execution, job termination verification, and job priority change popups.

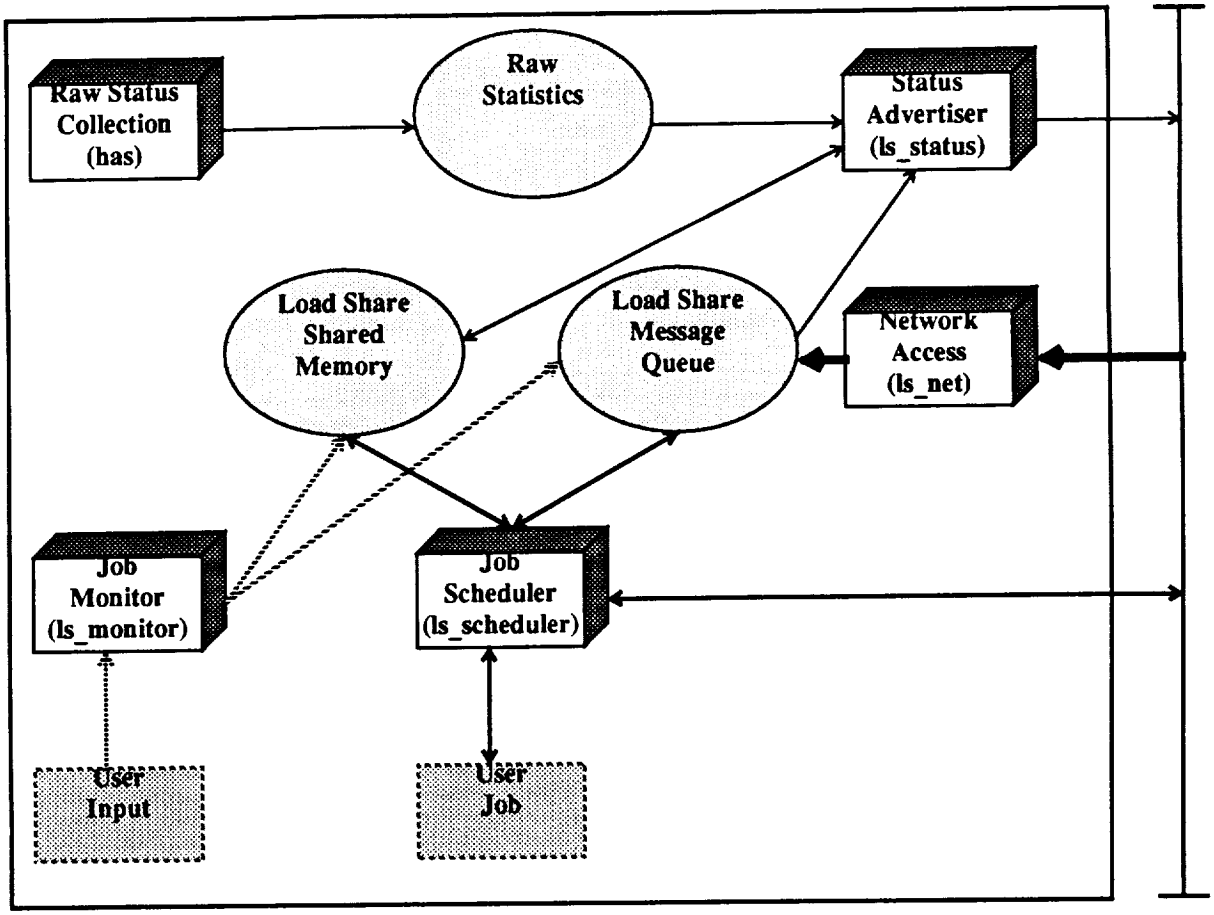

ls_monitor
File Jobs Hosts Miscellaneous
Help
Local Load
Low Value
High Value
100
100
50
100
30
70
00
0
50
Active Hosts
pegasus
andromeda
polaris
trident
taurus
Pid Pri Job Source Job Location User name Job Start Time Job Name
15279 000 pegasus taurus Dummy Tue Sep 25 13:56:11 1990 csh
01410 000 pegasus andromeda Dummy Tue Sep 25 13:59:46 1990 csh
06941 000 pegasus polaris Dummy Tue Sep 25 13:35:10 1990 csh
01413 020 pegasus andromeda Dummy Tue Sep 25 14:00:19 1990 csh
06944 000 pegasus polaris Dummy Tue Sep 25 13:35:28 1990 csh
15282 000 pegasus taurus Dummy Tue Sep 25 13:57:09 1990 csh
08248 000 pegasus trident Dummy Tue Sep 25 14:01:35 1990 csh
15285 010 pegasus taurus Dummy Tue Sep 25 13:57:34 1990 csh
01418 000 pegasus andromeda Dummy Tue Sep 25 14:02:02 1990 ico
06947 000 pegasus polaris Dummy Tue Sep 25 13:37:24 1990 ico
Status Messages
Job was added
Job was added
Job was added
Job was added
Job was added
Job was added
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Marks Xterm Windo
Ico
Ico
Ico
xterm


pegasus Parameters
File
Help
Local Load
Low Value
High Value
100
97
50
100
30
70
00
0
50
andromeda Parameters
File
Help
Local Load
Low Value
High Value
100
50
100
30
70
00
8
0
50
taurus Parameters
File
Help
Local Load
Low Value
High Value
100
50
100
30
70
00
0
0
50
polaris Parameters
File
Help
Local Load
Low Value
High Value
100
50
100
30
70
00
1
0
50
virgo Parameters
File
Help
Local Load
Low Value
High Value
100
50
100
30
70
00
0
0
50
trident Parameters
File
Help
Local Load
Low Value
High Value
100
50
100
30
70
00
31
0
50
Ico
xterm


Static Parameters
File
Help
Status Check Interval
1
Advertisement Interval
5
Heartbeat Interval
30
Host Check Interval
30
Host Timeout Value
90
Low Value Minimum
0
Low Value Maximum
50
High Value Minimum
50
High Value Maximum
100
Execute Job
Enter job and parameters
csh
xterm
No xterm
Cancel
Help
Kill Job_popup
Are you sure you want to kill this job
OK
Cancel
Help
Set Priority
Specify the desired priority
0
20
0
Ok
Cancel
Help
xterm

4.0 Data Flow Diagram

The following diagram illustrates the flow of data through the load sharing system:



Legend:

- Faint line - status data flow.
- Medium line - job control flow.
- Heavy line - network datagram input.
- Dashed line - user input and requests.

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( ls_init )
 *
 *  This program is a daemon process which is responsible initialization of the load
 *  sharing system. This process is responsible for the following:
 *
 *      o   Initialization of signals.
 *      o   Execution of the health and status (has) process.
 *      o   Creation of the shared memory segment used to share data.
 *      o   Creation of the "port" used to send and receive datagrams.
 *      o   Creation of the message queue used to buffer messages.
 *      o   Creation of the semaphore used to control access to shared memory.
 *      o   Initialization of load sharing system memory.
 *      o   Execution of all load sharing processes.
 *
 *  Once all initialization is complete, this process will enter a wait state in
 *  which it will respond to changes of state in any of its child processes (such
 *  as a termination). The processes spawned include:
 *
 *      o   ls_net - processes and queues incoming datagrams.
 *      o   ls_status - retrieves and broadcasts status data.
 *      o   ls_scheduler - schedules local and remote jobs.
 *      o   ls_monitor - provides all user interface for job execution, control
 *          and status display.
 *
 *  If a child dies, it will be respawned. This is necessary, as all processes are
 *  required for normal operation. The only normal termination for child processes
 *  will be initiated from this process. If this process receives a kill signal, it
 *  will terminate all child processes and clean up all resources. This procedure
 *  insures an orderly shutdown.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cleanup - kills all child processes and removes all resources.
 *  o   init_has - initializes the health and status (has) process.
 *  o   init_message_queue - initializes the message queue used by all processes.
 *  o   init_processes - executes all child processes.
 *  o   init_semaphore - initializes the semaphore.
 *  o   init_shared_memory - initializes the shared memory segment.
 *  o   init_shm_values - initializes the shared memory segment with all data.
 *  o   test_for_has - checks if the health and status (has) process is running.
 *  o   which_pid - converts a process id into an index into the process table.
 *
 *
 * COMMAND LINE EXECUTION:
 *
 *  ls_init
 *
 *
 * REQUIRED EXTERNAL RESOURCES:
 *
 *  o   The shared memory segment used to contain all load sharing data.
 *
 *  o   The TCP/IP port used for transmission of all datagrams. The port
 *      number is retrieved from shared memory.
 *
 *  o   The message queue used to queue up all messages and requests for the
 *      load sharing system.
 *
 *  o   The semaphore used to control access to shared memory.
 *
 *
 * FILES:
 *
 *  ./data/parms - contains non-default values for the local system.
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>
#include <stdio.h>
#include <ctype.h>
#include <ls.h>
#include <ls_init.h>
#include <configure.h>


struct resource_str     resources;

char                    process[] = "ls_init";

extern int  errno;


main ( argc, argv )

    int     argc;

    char    **argv;
{
    register int        i,
                        pid;

    static char         f[] = "ls_init";

    struct ls_shm_str   *ls_shm;

    extern void         shutdown  ( ),
                        terminate ( );

    D(printf("\nLS_INIT\n"));
/*
 *  Initialize each value in the resource structure to indicate that no value has been
 *  set yet. This allows the (cleanup) function to be called at any time and to perform
 *  the correct clean up actions.
 */

    for ( i = 0; i < MAX_LS_PROCESSES; i++ )
        resources.res_proc_pids[i] = -1;

    resources.res_has_shm_id = -1;
    resources.res_sys_shm_id = -1;
    resources.res_port       = -1;
```

```
    resources.res_mq_id       = -1;
    resources.res_sem_id      = -1;

/*
 *  Initialize to trap signals. If an abnormal signal occurs, the (terminate) function
 *  will be called. If a normal kill signal is received, the (shutdown) function will
 *  be called. In both cases, an orderly shutdown will be performed (cleanup will be
 *  called).
 */

    init_signals ( shutdown, terminate );

/*
 *  Call various functions to perform the actual initialization steps. This includes
 *
 *      Execution of the (has) process.
 *      Initialization of load sharing system memory.
 *      Creation of the "port" used to send and receive datagrams.
 *      Creation of the message queue used to buffer message.
 *      Creation of the semaphore used to control access to shared memory.
 *      Initialization of load sharing system memory with default values.
 *      Run all processes.
 *
 *  If any function fails, call (cleanup) to remove all allocated resources and
 *  terminate with nonzero error.
 */

    if ( ( resources.res_has_shm_id = init_has           (                    ) ) == -1
||
         ( resources.res_sys_shm_id = init_shared_memory ( &ls_shm            ) ) == -1
||
         ( resources.res_port       = net_create_port    (                    ) ) == -1
||
         ( resources.res_mq_id      = init_message_queue (                    ) ) == -1
||
         ( resources.res_sem_id     = init_semaphore     (                    ) ) == -1
||
                                      init_shm_values    ( ls_shm, &resources )   == -1
||
                                      init_processes     ( &resources, -1     )   == -1
) {
        cleanup ( );
        exit ( 1 );
    }

/*
 *  Loop infinitely waiting for death of any child process.
 */

    while ( 1 ) {

/*
 *      Loop infinitely waiting for death of any child process. Do a blocking (wait)
 *      for a child termination. Note that this could be done with a SIGCHLD signal
 *      and a separate function. I did it this way because it is a little clearer.
 */

        pid = wait ( (union wait *)0 );

/*
 *      A child has died, so first log a message (errno is set to zero as it does not
 *      reflect the reason for the death). Next, if the child process was a load
 *      sharing process, call (init_processes) to restart the appropriate process.
 *      Note that the (which_pid) function converts the process id to an index into
 *      the table used to keep track of processes.
 */

        err_log ( f, "Process died, restarting it", 0, 0 );
        send_status_info ( ls_shm, "System process died, restarting it" );

        if ( ( i = which_pid ( &resources, pid ) ) >= 0 ) {
            if ( init_processes ( &resources, i ) == -1 ) {
                cleanup ( );
                exit ( 1 );
            }

/*
 *  Otherwise (which_pid returned a -1), it was the health and status (has) process
 *  which died.  So call (init_has) to reinitialize it.
 */

        } else {
            if ( ( resources.res_has_shm_id = init_has ( ) ) == -1 ) {
                cleanup ( );
                exit ( 1 );
            }
        }
    }
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cleanup )
 *
 *  This function is responsible for cleaning up any resources created by this
 *  process. This includes child processes, shared memory, the network port, the
 *  message queue, and the semaphore.
 *
 *  Note that the health and status (has) process is not killed as this process
 *  should always be running.
 *****************************************************************************/

int cleanup ( )
{
    register int    i;

    static char     f[] = "cleanup";

    D(printf("\nCLEANUP\n"));
/*
 *  Remove all load sharing processes. This is done by sending a SIGTERM signal,
 *  which is trapped by each child and causes a normal termination sequence. Note that
 *  by not doing a wait, the processes are "zombies" until this process completes.
 */

    for ( i = 0; i < MAX_LS_PROCESSES; i++ )
        if ( resources.res_proc_pids[i] != -1 )
            kill ( resources.res_proc_pids[i], SIGTERM );

/*
 *  Sleep for a few seconds to be sure that processes are actually finished.
 *  Note that by not doing a wait, the processes are "zombies" until this process
 *  completes.
 */

    sleep ( 2 );

/*
 *  Delete the shared memory segment.
 */

    if ( resources.res_sys_shm_id != -1 )
        if ( shmctl ( resources.res_sys_shm_id, IPC_RMID, 0 ) == -1 )
            err_log ( f, "Could not delete shared memory", -1, errno );

/*
 *  Deallocate the datagram port.
 */

    if ( resources.res_port != -1 )
        net_close_port ( resources.res_port );

/*
 *  Delete the message queue.
 */

    if ( resources.res_mq_id != -1 )
        if ( msgctl ( resources.res_mq_id, IPC_RMID, 0 ) == -1 )
            err_log ( f, "Could not deallocate message queue", -1, errno );

/*
 *  Delete the semaphore.
 */

    if ( resources.res_sem_id != -1 )
        if ( semctl ( resources.res_sem_id, IPC_RMID, 0 ) == -1 )
            err_log ( f, "Could not deallocate semaphore", -1, errno );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_has )
 *
 *  This function is used to determine if the health and status (has) process is
 *  running. If not, this function will execute it. The (has) process is required
 *  to gather raw system statistics.
 *****************************************************************************/

int init_has ( )
{
    register int    pid,
                    shmid = -1,
                    trys  = 2;

    static char     f[] = "init_has";

    D(printf("\nINIT_HAS\n"));
/*
 *  Loop for (trys) number of tries in an attempt to determine if the (has) process
 *  is active. The (test_for_has) function will return the shared memory id if (has)
 *  is running.  Note that if (has) is not running, the first try fails and execution
 *  is attempted. If all goes well, the (testforhas) function will return a valid
 *  shared memory id on the second try.
 */

    while ( shmid == -1 && trys-- ) {
        if  ( ( shmid = test_for_has ( ) ) == 0 ) {

/*
 *          If (has) is not active, fork and execute the has process with the
 *          default set of statistic parameters. Upon return from the fork in the
 *          parent process, sleep for a few seconds to allow the (has) process to
 *          initialize. Note that the error call is executed in the child, as it
 *          is the only one who knows the reason for the error.
 */

            if ( ( pid = fork ( ) ) == 0 ) {
                execlp ( PATH_HAS, PATH_HAS,
                        "-p", PROC_STAT_CYCLE,     "-s", SYSINFO_STAT_CYCLE,
                        "-v", VMMETER_STAT_CYCLE, "-f", FORK_STAT_CYCLE,
                        "-n", NET_STAT_CYCLE,      "-F", FILESYS_STAT_CYCLE,
                        "-c", CYCLE_STAT_CYCLE,
                        (char *)0);
                err_log ( f,  "Could not exec the health and status process", -1, errno
);
                exit ( 1 );
            } else if ( pid > 0 )
                sleep ( 2 );
            else
                return ( err_log ( f, "Unable to fork process", -1, errno ) );
        }
    }

/*
 *  If the attempts failed, log a fatal error. Note that (errno) is zeroed out
 *  because it will not reflect the cause.
 */

    if ( shmid == -1 ) {
        return ( err_log ( f, "Unable to execute or attach to has", -1, 0 ) );
    }

/*
 *  Return the shared memory ID.
 */

    return ( shmid );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_message_queue )
 *
 *  This function is responsible for initializing the message queue used by the load
 *  sharing system to queue up messages.
 *****************************************************************************/

init_message_queue ( )
{
    register int    mq;

    static char     f[] = "init_message_queue";

    D(printf("\nIINIT_MESSAGE_QUEUE\n"));
/*
 *  Try to create the message queue. This call will fail if the message queue
 *  already exists (due to the IPC_EXCL flag).
 */

    if ( ( mq = msgget ( LS_MQ_KEY, IPC_CREAT | IPC_EXCL | 0600 ) ) == -1 ) {

/*
 *      Create failed. If this is due to the message queue already existing, then
 *      attempt to first delete the message queue and then create a new one. If either
 *      operation fails, log and return an error.
 */

        if ( errno == EEXIST ) {
            if ( msgctl ( mq, IPC_RMID, 0 ) == -1 )
                return ( err_log ( f, "Could not delete message queue", -1, errno ) );
            if ( ( mq = msgget ( LS_MQ_KEY, IPC_CREAT | 0600 ) ) == -1 )
                return ( err_log ( f, "Could not create message queue", -1, errno ) );

/*
 *      Otherwise the create failed due to some other problem.
 */

        } else
            return ( err_log ( f, "Could not create message queue", -1, errno ) );
    }

/*
 *  Normal return of the newly created message queue id.
 */

    return ( mq );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_processes )
 *
 *  This function is responsible for execution of all child processes which make up
 *  the load sharing system. This function executes all processes and then waits to
 *  to verify that all have been run successfully.
 *
 *  This function can also be called to re-execute a single process. This operation
 *  is indicated by setting the (process) parameter to a value greater than or
 *  equal to zero. Such a value corresponds to the index of the desired process.
 *****************************************************************************/

int init_processes ( resources, process )

    struct resource_str *resources;

    int                 process;
{
    register int    error = 0,
                    i,
                    max,
                    p_index,
                    pid;

    static char     f[] = "init_processes";

    char            string[MAX_STRING + 1];

    D(printf("\nINIT_PROCESSES\n"));
/*
 *  Set values which control the operation of the following loops. If (process) is
 *  a negative value, set values to cause initialization of all processes. Otherwise,
 *  set up to only execute the corresponding process (process is an index into the
 *  process table).
 */

    i   = ( process >= 0  ) ? process     : 0;
    max = ( process >= 0  ) ? process + 1 : MAX_LS_PROCESSES;

/*
 *  Perform a loop which forks and exec's each requested process. Note that if
 *  the child cannot exec the process, it will log an error. This is necessary as
 *  only it has access to the appropriate value of (errno).
 */

    for ( ; i < max; i++ )
        if ( ( resources->res_proc_pids[i] = pid = fork ( ) ) == 0 ) {
            execl ( process_paths[i], process_paths[i], (char *)0 );
            sprintf ( string, "Could not exec %s", process_paths[i] );
            err_log ( f, string, -1, errno );
            exit ( 1 );
        } else if ( pid < 0 )
            return ( err_log ( f, "Could not fork process", -1, errno ) );

/*
 *  Sleep to allow all processes to initialize.
 */

    sleep ( 5 );

/*
 *  Reset the value of (i) to handle the requested process(es) in the next loop.
 */
```

```
    i = ( process >= 0 ) ? process : 0;

/*
 *  For each of the processes to be run, do a non-blocking wait to determine if the
 *  process actually ran.
 */

    for ( ; i < max; i++ ) {
        pid = wait3 ( (union wait *)0, WNOHANG, (struct rusage *)0 );

/*
 *      If the wait returns a non-zero pid, then it is the pid of a terminated
 *      child.  In this case, determine which process the pid is associated and
 *      log a descriptive error. Note that (errno) is zeroed as it has no meaning.
 */
        if ( pid ) {
            errno = 0;
            if ( ( p_index = which_pid ( resources, pid ) ) >= 0 ) {
                sprintf ( string, "Process %s terminated", process_paths[p_index] );
                err_log ( f, string, error = -1, errno );

/*
 *          The (which_pid) function returned a negative value, which indicates
 *          that it was the (has) process which terminated.
 */

            } else
                err_log ( f, "The has process terminated", error = -1 );
        }
    }

/*
 *  Return error status.
 */

    return ( error );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_semaphore )
 *
 *  This function is responsible for initializing the semaphore which is used to
 *  control access to the shared memory segment.
 *****************************************************************************/

init_semaphore ( )
{
    register int    sem;

    static char     f[] = "init_semaphore";

    D(printf("\nINIT_SEMAPHORE\n"));
/*
 *  Try to create the semaphore. This call will fail if the message queue already
 *  exists (due to the IPC_EXCL flag).
 */

    if ( ( sem = semget ( LS_SEM_KEY, 1, IPC_CREAT | IPC_EXCL | 0600 ) ) == -1 ) {

/*
 *      Create failed. If this is due to the semaphore already existing, then attempt
 *      to first delete the semaphore and then create a new one. If either operation
 *      fails, log and return an error.
 */

        if ( errno == EEXIST ) {
            if ( semctl ( sem, IPC_RMID, 0 ) == -1 )
                return ( err_log ( f, "Could not delete semaphore", -1, errno ) );
            if ( ( sem = semget ( LS_SEM_KEY, 1, IPC_CREAT | 0666 ) ) == -1 )
                return ( err_log ( f, "Could not create semaphore", -1, errno ) );
        } else
            return ( err_log ( f, "Could not create semaphore", -1, errno ) );
    }

/*
 *  Unlock the semaphore to allow it to be used by other applications.
 */

    sem_unlock ( sem );

/*
 *  Return the semaphore id.
 */

    return ( sem );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_shared_memory )
 *
 *  This function is responsible for initializing the shared memory segment used by
 *  all load sharing processes.
 *****************************************************************************/

int init_shared_memory ( shm_addr )

    struct ls_shm_ptr   **shm_addr;
{
    static char f[] = "init_shared_memory";

    int         shm_id;

    extern char *shmat ( );

    D(printf("\nINIT_SHARED_MEMORY\n"));
/*
 *  Try to create the shared memory segment. This call will fail if the shared memory
 *  segment already exists (due to the IPC_EXCL flag).
 */

    if ( ( shm_id = shmget ( LS_SHM_KEY, sizeof ( struct ls_shm_str ),
                             ( IPC_CREAT | IPC_EXCL | 0600 ) ) ) == -1 ) {

/*
 *      Create failed. If this is due to the shared memory segment already existing,
 *      then attempt to first delete the shared memory segment and then create a
 *      new one. If either operation fails, log and return an error.
 */

        if ( errno == EEXIST ) {
            if ( shmctl ( shm_id, IPC_RMID, 0 ) == -1 )
                return ( err_log ( f, "Can't delete shared memory", -1, errno ) );

            if ( ( shm_id = shmget ( LS_SHM_KEY, sizeof ( struct ls_shm_str ),
                                     ( IPC_CREAT | 0666 ) ) ) == -1 )
                return ( err_log ( f, "Can't allocate shared memory", -1, errno ) );
        } else
            return ( err_log ( f, "Can't allocate shared memory", -1, errno ) );
    }

/*
 *  Attach to shared memory and save address in the (shm_addr) parameter.
 */

    if ( (int)( *shm_addr = (struct ls_shm_ptr *)shmat ( shm_id, 0, 0 ) ) == -1 )
        return ( err_log ( f, "Could not attach to shared memory", -1, errno ) );

/*
 *  Return the shared memory ID.
 */

    return ( shm_id );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_shm_values )
 *
 *  This function initializes the shared memory segment. This includes host and job
 *  table initialization, saving of system resource values (ports, message queue
 *  IDs), and default values.
 *****************************************************************************/

int init_shm_values ( ptr, resources )

    struct ls_shm_str   *ptr;

    struct resource_str *resources;
{
    register int    i;

    static char     f[] = "init_shm_values";

    char            string    [MAX_STRING + 1],
                    ident     [MAX_STRING + 1],
                    parameter [MAX_STRING + 1];

    FILE            *fp;

    D(printf("\nINIT_SHM_VALUES\n"));
/*
 *  Update shared memory with the port, the message queue id, and the semaphore from
 *  the resources structure.
 */

    ptr->ls_port   = resources->res_port;
    ptr->ls_mq_id  = resources->res_mq_id;
    ptr->ls_sem_id = resources->res_sem_id;

/*
 *  Initialize the counters which indicate the number of hosts.  Note that there is
 *  always at least one host because there is a local host.
 */

    ptr->ls_num_hosts = 1;
    ptr->ls_num_jobs  = 0;

/*
 *  Set the time of the current update to the host and job tables.
 */

    ptr->ls_last_host_upd = time ( (time_t *)0 );
    ptr->ls_last_job_upd  = time ( (time_t *)0 );

/*
 *  Initialize all process ids in the job table to -1 to indicate that no job is
 *  active in that entry. The job array is handled like this due to its more
 *  dynamic nature.
 */

    for ( i = 0; i < MAX_JOBS; i++ ) {
        ptr->ls_jobs[i].job_type = TYPE_JOB_RESP;
        ptr->ls_jobs[i].job_pid  = -1;
    }

/*
 *  Initialize the type value in each host structure so that they can be sent as
 *  messages and datagrams.
```

```
 */

    for ( i = 0; i < MAX_HOSTS; i++ )
        ptr->ls_hosts[i].host_type = TYPE_STATUS;

/*
 *  Initialize the system wide defaults which the user cannot change. These values
 *  are the same for all workstations.
 */

    ptr->ls_status_int      = PARM_STATUS_INT;
    ptr->ls_advert_int      = PARM_ADVERT_INT;
    ptr->ls_heartbeat_int   = PARM_HEARTBEAT_INT;
    ptr->ls_check_host_int  = PARM_CHECK_HOST_INT;
    ptr->ls_host_timeout    = PARM_HOST_TIMEOUT;
    ptr->ls_low_value_min   = PARM_LOW_VALUE_MIN;
    ptr->ls_low_value_max   = PARM_LOW_VALUE_MAX;
    ptr->ls_high_value_min  = PARM_HIGH_VALUE_MIN;
    ptr->ls_high_value_max  = PARM_HIGH_VALUE_MAX;

/*
 *  Initialize the first entry in the host table to information for the current
 *  host. Note that these are pure defaults which can be overridded by values in a
 *  configuration file.
 */

    gethostname ( ptr->ls_hosts[0].host_name, MAX_HOST_NAME );

    ptr->ls_hosts[0].host_cpu_factor    = PARM_HOST_CPU_FACTOR;
    ptr->ls_hosts[0].host_disk_factor   = PARM_HOST_DISK_FACTOR;
    ptr->ls_hosts[0].host_risc_cisc     = PARM_HOST_RISC_CISC;
    ptr->ls_hosts[0].host_high_value    = PARM_HOST_HIGH_VALUE;
    ptr->ls_hosts[0].host_low_value     = PARM_HOST_LOW_VALUE;

    ptr->ls_hosts[0].host_memory_size   = PARM_HOST_MEMORY_SIZE;

/*
 *  Initialize the array of load history values. These must be initialized, as this
 *  structure acts as a circular queue. This structure is updated each time status
 *  is collected and averages any time the system load is computed.
 */

    for ( i = 0; i < PARM_ADVERT_INT; i++ ) {
        ptr->ls_raw_stats[i].rs_users  = -1;
        ptr->ls_raw_stats[i].rs_procs  = -1;
        ptr->ls_raw_stats[i].rs_cpu    = -1;
        ptr->ls_raw_stats[i].rs_memory = -1;
    }
    ptr->ls_stats_ptr = PARM_ADVERT_INT - 1;

/*
 *  Initialize the defaults for the local host if a config file exists. Note that
 *  it is not a fatal error if this file does not exist.
 */

    if ( ( fp = fopen ( PATH_HOST_PARMS, "r" ) ) == NULL ) {
        err_log ( f, "Could not open parameters file, using defaults", 0, 0 );
        return ( 0 );
    }

/*
 *  Read and parse each line from the configuration file. Ignore any line which is
 *  blank or starts with a '#' character.
 */

    while ( fgets ( string, MAX_STRING, fp ) != NULL ) {
        if ( *string != '#' && *string != '\n' ) {

/*
 *          Parse out the keyword and parameter from the current record. Shift each
 *          character to lower case.
 */

            sscanf ( string, "%s %s", ident, parameter );
            if ( *ident && *parameter ) {
                for ( i = 0; ident[i]; i++ )
                    if ( isupper ( ident[i] ) )
                        ident[i] = tolower ( ident[i] );

/*
 *              Based on the keyword in the record, extract the corresponding
 *              value and set in shared memory.
 */

                if      ( strcmp ( ident, "host_cpu_factor" ) == 0 )
                    ptr->ls_hosts[0].host_cpu_factor = atoi ( parameter );
                else if ( strcmp ( ident, "host_disk_factor" ) == 0 )
                    ptr->ls_hosts[0].host_disk_factor = atoi ( parameter );
                else if ( strcmp ( ident, "host_memory_size" ) == 0 )
                    ptr->ls_hosts[0].host_memory_size = atoi ( parameter );
                else if ( strcmp ( ident, "host_risc_cisc" ) == 0 )
                    ptr->ls_hosts[0].host_risc_cisc = atoi ( parameter );
                else if ( strcmp ( ident, "host_high_value" ) == 0 )
                    ptr->ls_hosts[0].host_high_value = atoi ( parameter );
                else if ( strcmp ( ident, "host_low_value" ) == 0 )
                    ptr->ls_hosts[0].host_low_value = atoi ( parameter );
                else
                    err_log ( f, "Invalid keyword in configuration file", 0, 0 );

            }
        }
    }

/*
 *  Close the parameters file.
 */

    fclose ( fp );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( test_for_has )
 *
 *  This function is used to check if the health and status (has) process is running
 *  and has initialized the appropriate shared memory segment. If so, this function
 *  will return the shared memory id of this segment.
 *****************************************************************************/

int test_for_has ( )
{
    static char     f[] = "test_for_has";

    struct shmid_ds status_buffer;

    int             shm_id;

    D(printf("\nTEST_FOR_HAS\n"));
/*
 *  See if a shared memory segment already exists and is readable by the current
 *  process. If not, return.
 */

    if ( ( shm_id = shmget ( SHARED_MEM_KEY, 0, 0444 ) ) == -1 ) {
        if ( errno == EACCES )
            return ( err_log ( f, "has SM exists, but no permission", 0, errno ) );
        else
            return ( 0 );
    }

/*
 *  The shared memory segment exists with read permissions, so get a status
 *  buffer to determine if the (has) process is attached and active. If this
 *  status operation fails, return.
 */

    if ( ( shmctl ( shm_id, IPC_STAT, &status_buffer ) ) == -1 )
        return ( err_log ( f, "Can't status shared memory id", 0, errno ) );

/*
 *  Test to see if the creating process ID is still alive (by sending a zero
 *  signal). Return zero if this fails (indicating that has is dead).
 */

    if ( kill ( status_buffer.shm_cpid, 0 ) == -1 && errno == ESRCH )
        return ( err_log ( f, "SM exists, but has appears to be dead", 0, errno ) );

/*
 *  The shared memory segment exists and (has) is active. Return the shared
 *  memory segment id.
 */

    return ( shm_id );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( which_pid )
 *
 *  This function is called to convert process ID into an index into the child
 *  process table. If a match is found, the index is returned; otherwise, a -1 is
 *  returned.
 *****************************************************************************/

int which_pid ( resources, pid )

    struct resource_str *resources;

    int                 pid;
{
    register int    i;

    D(printf("\nWHICH_PID\n"));
/*
 *  Scan the structure containing process pids if a match is found, return the
 *  matching index.
 */

    for ( i = 0; i < MAX_LS_PROCESSES; i++ )
        if ( pid == resources->res_proc_pids[i] )
            return ( i );

/*
 *  Return a -1, indicating that no match was found. This normally indicates that this
 *  function was called with the pid of (has).
 */

    return ( -1 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: (ls_lib)
 *
 *  This file contais the general library for the load sharing system.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      attach_shared_memory
 *      err_log
 *      init_signals
 *      net_close_port
 *      net_create_port
 *      net_get_datagram
 *      net_send_datagram
 *      sem_lock
 *      sem_unlock
 *      send_status_info
 *      shutdown
 *      terminate
 *
 *
 * REQUIRED EXTERNAL RESOURCES:
 *
 *  o   The shared memory segment used to contain all load sharing data.
 *
 *  o   The TCP/IP port used for transmission of all datagrams. The port
 *      number is retrieved from shared memory.
 *
 *  o   The message queue used to queue up all messages and requests for the
 *      load sharing system.
 *
 *  o   The semaphore used to control access to shared memory.
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/msg.h>
#include <sys/sem.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <ls.h>


extern char process[];
extern int  errno;
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( attach_shared_memory )
 *
 *  This function attaches the current process to the load sharing shared memory
 *  segment.
 *****************************************************************************/

int attach_shared_memory ( shm_addr )

    struct ls_shm_ptr   **shm_addr;
{
    static char f[] = "attach_shared_memory";

    int         shm_id;

    extern char *shmat();

    D(printf("\nATTACH_SHARED_MEMORY\n"));
/*
 *  Retrieve the shared memory id associated with the system shared memory.
 */

    if ( ( shm_id = shmget ( LS_SHM_KEY, 0, 0 ) ) == -1 )
        return ( err_log ( f, "Could not get shared memory id", -1, errno ) );

/*
 *  Attach to the shared memory and set address.
 */

    if ( (int)( *shm_addr = (struct ls_shm_ptr *)shmat ( shm_id, 0, 0 ) ) == -1 )
        return ( err_log ( f, "Could not attach to shared memory", -1, errno ) );

/*
 *  Return the shared memory ID.
 */

    return ( shm_id );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( err_log )
 *
 *  This function logs an error. The error includes the process, the function, a
 *  descriptive string, and if applicable, the errno.
 *****************************************************************************/

int err_log ( func, string, error, errno )

    char    *func,
            *string;

    int     error,
            errno;
{
    extern char *sys_errlist[];

    D(printf("\nERR_LOG\n"));
/*
 *  Output string indicating severity of the error.
 */

    if ( error == -1 )
        printf ( "FATAL - "    );
    else
        printf ( "NONFATAL - " );

/*
 *  Output the actual error string. It will contain the process name (external), the
 *  function encountering the error, a descriptive error string, and if (errno) is
 *  set, a description of the system error.
 */

    if ( errno )
        printf ( "%s->%s: %s - Cause: %s\n", process, func, string, sys_errlist[errno] )
;
    else
        printf ( "%s->%s: %s\n",             process, func, string                     )
;

/*
 *  Return the error to allow this function to be easily used inside a (return)
 *  statement.
 */

    return ( error );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( init_signals )
 *
 *  This function sets up the proper signal processing for the current process.
 *****************************************************************************/

int init_signals ( shutdown_proc, terminate_proc )

    void    (*shutdown_proc) (),
            (*terminate_proc)();
{
    static char f[] = "init_signals";

    D(printf("\nINIT_SIGNALS\n"));
/*
 *  Catch the signal which is used to properly terminate the load sharing system. The
 *  (shutdown_ls) function does an orderly cleanup and then exits with a status of 0.
 */

    signal ( SIGTERM, shutdown_proc );

/*
 *  Catch all signals which indicate a fatal error occurred.  The (terminate_ls) outputs
 *  the signal which occurred, does an orderly cleanup, and then exits with a status of 1
.
 */

    signal ( SIGQUIT, terminate_proc );
    signal ( SIGILL,  terminate_proc );
    signal ( SIGFPE,  terminate_proc );
    signal ( SIGBUS,  terminate_proc );
    signal ( SIGSEGV, terminate_proc );
    signal ( SIGSYS,  terminate_proc );

/*
 *  Ignore software interrupts (Control-C) and hangup (signal 1).
 */

    signal ( SIGINT, SIG_IGN );
    signal ( SIGHUP, SIG_IGN );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( net_close_port )
 *
 *  This function closes a network port (which is actually a socket).
 *****************************************************************************/

int net_close_port ( port )
{

    static char         f[] = "net_close_port";

    D(printf("\nINET_CLOSE_PORT\n"));
/*
 *  Close the port (the socket).
 */

    return ( close ( port ) );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( net_create_port )
 *
 *  This function creates a network port across which datagrams can be sent.
 *****************************************************************************/

int net_create_port ( )
{

    static char         f[] = "net_create_port";

    register int        sock;

    struct sockaddr_in  name;

    D(printf("\nNET_CREATE_PORT\n"));
/*
 *  Create the port (socket).
 */

    if ( ( sock = socket ( AF_INET, SOCK_DGRAM, 0 ) ) < 0 )
        return ( err_log ( f, "Could not create INTERNET datagram socket", -1, errno ) );

/*
 *  Initialize the name which to bind.
 */

    name.sin_family      = AF_INET;
    name.sin_addr.s_addr = INADDR_ANY;
    name.sin_port        = NET_PORT;

    if ( bind ( sock, (struct sockaddr *)&name, sizeof ( name ) ) < 0 )
        return ( err_log ( f, "Could not bind INTERNET datagram socket", -1, errno ) );

/*
 *  Return the port (socket).
 */

    return ( sock );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( net_get_datagram )
 *
 *  This function waits on a port (a socket) for arrival of a datagram. This is a
 *  blocking wait.
 *****************************************************************************/

int net_get_datagram ( socket, message )

    int                     socket;

    struct ls_message_str   *message;
{
    static char     f[] = "net_get_datagram";

    register int    length;

    D(printf("\nNET_GET_DATAGRAM\n"));
/*
 *  Wait until a datagram arrives at the socket.
 */

    if ( ( length = read ( socket, message, MAX_MESSAGE ) ) < 0 )
        return ( err_log ( f, "Error reading INTERNET datagram socket", -1, errno ) );

/*
 *  Return the length.
 */

    return ( length );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( net_send_datagram )
 *
 *  This function sends a datagram across the network.
 *****************************************************************************/

int net_send_datagram ( hostname, socket, message, length )

    char                    *hostname;

    struct ls_message_str   *message;

    int                     socket,
                            length;
{
    static char         f[] = "net_send_datagram";

    struct hostent      *hp;

    struct sockaddr_in  name;

    char                s[MAX_STRING + 1];

    D(printf("\nNET_SEND_DATAGRAM\n"));
/*
 *  If no hostname was passed, assume a broadcast datagram.
 */

    if ( hostname == 0 )
        hostname = BROADCAST_NAME;

/*
 *  Initialize the address for the host name.
 */

    if ( ( hp = gethostbyname ( hostname ) ) == 0 ) {
        sprintf ( s, "Unable to determine address for (%s)", hostname );
        return ( err_log ( f, s, -1, errno ) );
    }

/*
 *  Set up the actual name structure to send the datagram to.
 */

    bcopy ( (char *)hp->h_addr, (char *)&name.sin_addr, hp->h_length );
    name.sin_family = AF_INET;
    name.sin_port   = NET_PORT;

/*
 *  Actually send the datagram.
 */

    if ( sendto ( socket, (char *)message, length, 0,
                  (struct sockaddr *)&name, sizeof ( name ) ) < 0 )
        return ( err_log ( f, "Error writing INTERNET datagram socket", -1, errno ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( sem_lock )
 *
 *  This function locks a semaphore.
 *****************************************************************************/

sem_lock ( sem )

    int sem;
{
    static char     f[] = "sem_lock";

    struct sembuf   sops;

    D(printf("\nSEM_LOCK\n"));
/*
 *  Initialize semaphore parameters and attempt to lock the semaphore.
 */

    sops.sem_num = 0;
    sops.sem_op  = -1;
    sops.sem_flg = 0;

    if ( semop ( sem, &sops, 1 ) == -1 )
        return ( err_log ( f, "Could not lock semaphore", 0, errno ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( sem_unlock )
 *
 *  This function is runs a job.
 *****************************************************************************/

sem_unlock ( sem )

    int sem;
{
    static char     f[] = "sem_unlock";

    struct sembuf   sops;

    D(printf("\nSEM_UNLOCK\n"));
/*
 *  Initialize semaphore parameters and attempt to unlock the semaphore.
 */

    sops.sem_num = 0;
    sops.sem_op  = 1;
    sops.sem_flg = 0;

    if ( semop ( sem, &sops, 1 ) == -1 )
        return ( err_log ( f, "Could not unlock semaphore", 0, errno ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( send_status_info )
 *
 *  This function puts an information message on the message queue to be removed
 *  and displayed by (ls_monitor).
 *****************************************************************************/

int send_status_info ( ls, string )

    struct ls_shm_str   *ls;

    char                            *string;
{
    static struct ls_message_str    message = { TYPE_INFO, "" };

    static char                     f[] = "send_status_info";

    D(printf("\nSEND_STATUS_INFO\n"));
/*
 *  Copy the information message into the message buffer.
 */

    strcpy ( message.msg_data, string );

/*
 *  Send the message.
 */

    if ( msgsnd ( ls->ls_mq_id, (struct msgbuf *)&message,
                  sizeof ( int ) + strlen ( string ), 0 ) == -1 )
        return ( err_log ( f, "Could not write to message queue", -1, errno ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( shutdown )
 *
 *  This function is called when the process is to perform a normal shutdown.
 *****************************************************************************/

void shutdown ( sig, code, scp, addr )

    int                 sig,
                        code;

    struct sigcontext   *scp;

    char                *addr;
{
    static char f[] = "shutdown";

    D(printf("\nSHUTDOWN\n"));
/*
 *  Report that the SIGTERM signal was received, clean up, and exit with a zero (normal)
 *  error status.
 */

    err_log ( f, "Received a SIGTERM signal, performing normal termination", 0, 0 );
    cleanup ( );

/*
 *  Normal exit.
 */

    exit ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( terminate )
 *
 *  This function is called when the process is to perform an abnormal shutdown.
 *  This will occur when certain signals are trapped.
 *****************************************************************************/

void terminate ( sig, code, scp, addr )

    int                 sig,
                        code;

    struct sigcontext   *scp;

    char                *addr;
{
    static char f[] = "terminate";

    char        string[MAX_STRING + 1];

    D(printf("\nTERMINATE\n"));
/*
 *  Report the signal that was received, clean up, and exit with a non-zero (error)
 *  error status.
 */

    sprintf ( string, "Received a signal %d, performing abnormal termination", sig );
    err_log ( f, string, -1, 0 );
    cleanup ( );

/*
 *  Exit with an error.
 */

    exit ( 1 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: (ls_net)
 *
 *  This program is a daemon process which is responsible for processing all in-
 *  coming network datagrams. This process waits on the load sharing port and
 *  when a datagram arrives, outputs the datagram on the load sharing message
 *  queue. This process is kept very simple to insure that it can effectively
 *  process all incoming datagrams.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *  o   cleanup - cleans up any resources created by this process.
 *
 *
 * COMMAND LINE EXECUTION:
 *
 *  ls_net
 *
 *
 * REQUIRED EXTERNAL RESOURCES:
 *
 *  o   The shared memory segment used to contain all load sharing data.
 *
 *  o   The TCP/IP port used for transmission of all datagrams. The port
 *      number is retrieved from shared memory.
 *
 *  o   The message queue used to queue up all messages and requests for the
 *      load sharing system.
 *
 *  o   The semaphore used to control access to shared memory.
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <ls.h>
#include <ls_net.h>


struct resource_str resources;

char                process[] = "ls_net";

extern int          errno;


main ( argc, argv )

    int     argc;

    char    **argv;
{
    static char             f[] = "ls_net";

    register int            length;

    struct ls_shm_str       *ls;

    struct ls_message_str   message;

    extern void             shutdown  ( ),
                            terminate ( );

    D(printf("\nLS_NET\n"));
/*
 *  Initialize to trap signals. If an abnormal signal occurs, the (terminate) function
 *  will be called. If a normal kill signal is received, the (shutdown) function will
 *  be called. In both cases, an orderly shutdown will be performed (cleanup will be
 *  called).
 */

    init_signals ( shutdown, terminate );

/*
 *  Attach to the system shared memory segment. (ls) will point to the shared memory
 *  segment.
 */

    if ( attach_shared_memory ( &ls ) == -1 ) {
        cleanup ( );
        exit ( 1 );
    }

/*
 *  Begin a loop which never terminates. This function terminates via the signal
 *  mechanisms previously described.
 */

    while ( 1 ) {

/*
 *      Wait on the load sharing port for arrival of a datagram. This includes both
 *      broadcast and point to point datagrams.
 */

        if ( ( length = net_get_datagram ( ls->ls_port, &message ) ) == -1 ) {
            cleanup ( );
            exit ( 1 );
        }

/*
 *      Write the received datagram intact onto the message queue. Note that the
 *      (send_message) library function is not used because it assumes that the
 *      message type and data need to be copied into a new, contiguous area (the
 *      most common case). In this case, the datagram structure corresponds to the
 *      message structure and there can be send directly.
 */

        if ( msgsnd ( ls->ls_mq_id, (struct msgbuf *)&message, length, 0 ) == -1 ) {
            err_log ( f, "Could not write to message queue", -1, errno );
            cleanup ( );
            exit ( 1 );
        }
    }
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: (cleanup)
 *
 *  This function is responsible for cleaning up any resources created by this
 *  process. Note that even though this process does not initialize any resources,
 *  this mechanism is left in place for future growth and to be consistent with
 *  other processes.
 *****************************************************************************/


int cleanup ( )
{
    static char f[] = "cleanup";

    D(printf("\nCLEANUP\n"));
/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: (ls_scheduler)
 *
 *  This is the main scheduler for the load sharing system.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      ls_scheduler
 *      add_job
 *      best_host
 *      child_termination
 *      cleanup
 *      process_job
 *      remove_job
 *      schedule_local_job
 *      set_priority_on_job
 *      terminate_job
 *      update_job
 *
 *
 * COMMAND LINE EXECUTION:
 *
 *  ls_scheduler
 *
 *
 * REQUIRED EXTERNAL RESOURCES:
 *
 *  o   The shared memory segment used to contain all load sharing data.
 *
 *  o   The TCP/IP port used for transmission of all datagrams. The port
 *      number is retrieved from shared memory.
 *
 *  o   The message queue used to queue up all messages and requests for the
 *      load sharing system.
 *
 *  o   The semaphore used to control access to the shared memory segment.
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <sys/types.h>
#include <sys/wait.h>
#include <sys/timeb.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <ls.h>
#include <ls_scheduler.h>


struct resource_str resources;

struct ls_shm_str   *ls;

char                process[] = "ls_scheduler",
                    *envp[100];

extern int          errno;


main ( argc, argv )

    int     argc;

    char    **argv;
{
    register int            length;

    static char             f[] = "ls_scheduler";

    struct ls_message_str   message;

    extern void             child_termination ( ),
                            shutdown          ( ),
                            terminate         ( );

    D(printf("\nLS_SCHEDULER\n"));
/*
 *  Initialize to trap signals. If an abnormal signal occurs, the (terminate) function
 *  will be called. If a normal kill signal is received, the (shutdown) function will
 *  be called. In both cases, an orderly shutdown will be performed (cleanup will be
 *  called).
 */

    init_signals ( shutdown, terminate );

/*
 *  Set up to trap changes in the state of any child processes. This allows this
 *  process to respond to terminated children.
 */

    signal ( SIGCHLD, child_termination );

/*
 *  Attach to the system shared memory segment. (ls) will point to the shared memory
 *  segment.
 */

    if ( attach_shared_memory ( &ls ) == -1 ) {
        cleanup ( );
        exit ( 1 );
    }

/*
 *  Begin a loop which never terminates. This function terminates via the signal
 *  mechanisms previously described.
 */

    while ( 1 ) {

/*
 *      Wait on the message queue for the next schedule request to arrive.
 */

    errno = EINTR;
    while ( errno == EINTR ) {
        errno = 0;
        if ( ( length = msgrcv ( ls->ls_mq_id, (struct msgbuf *)&message, MAX_MESSAGE,
```

```
                        -(TYPE_STATUS - 1), 0 ) ) == -1 ) {
            if ( errno != EINTR ) {
                err_log ( f, "Could not read schedule requests from queue", -1, errno );
                cleanup ( );
                exit ( 1 );
            }
        }
    }

/*
 *      Act based on the type of message received.
 */

        switch ( message.msg_type ) {

        case    TYPE_SCHEDULE:
            D(printf ( "SCHEDULE REQUEST\n" ));
            process_job ( ls, (struct ls_job_sched_str *)&message, length );
            break;

        case    TYPE_KILL_JOB:
            D(printf ( "TERMINATE JOB REQUEST\n" ));
            terminate_job ( ls, (struct ls_job_kill_str *)&message );
            break;

        case    TYPE_SET_PRIORITY:
            D(printf ( "PRIORITY CHANGE REQUEST\n" ));
            set_priority_on_job ( ls, (struct ls_job_pri_str *)&message );
            break;

        case    TYPE_JOB_RESP:
            D(printf ( "REMOTE JOB SCHEDULED RESPONSE\n" ));
            add_job ( ls, 1, &message, 0 );
            break;

        case    TYPE_DEAD_RESP:
            D(printf ( "REMOTE JOB TERMINATION RESPONSE\n" ));
            remove_job ( ls, ((struct ls_job_kill_resp_str *)&message)->jk_pid );
            break;

        case    TYPE_PRIORITY_RESP:
            D(printf ( "REMOTE JOB PRIORITY RESPONSE\n" ));
            update_job ( ls, ((struct ls_job_pri_resp_str *)&message)->jp_pid,
                             ((struct ls_job_pri_resp_str *)&message)->jp_priority );
            break;


        }
    }
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( add_job )
 *
 *  This function adds a job to the job table.
 *****************************************************************************/

int add_job ( ls, flag, data, pid )

    struct ls_shm_str       *ls;

    struct ls_message_str   *data;

    int                     flag,
                            pid;
{
    register int            i;

    static char             f[] = "add_job";

    struct ls_job_str       *ptr;

    struct ls_job_sched_str *job_s;

    char                    *string[100];

    D(printf("\nADD_JOB\n"));
/*
 *  Find the first available entry in the job table to insert the job.
 */

    for ( i = 0; i < MAX_JOBS; i++ )
        if ( ls->ls_jobs[i].job_pid == -1 )
            break;

/*
 *  Output an error if no room in job table.
 */

    if ( i == MAX_JOBS ) {
        err_log ( f, "Job table full", 0, 0 );
        send_status_info ( ls, "Job table full, job not tracked" );
        return ( -1 );
    }

/*
 *  Copy all the job information into the job table.
 */

    ptr = &ls->ls_jobs[i];

    if ( flag ) {
        memcpy ( (char *)ptr, (char *)data, sizeof ( struct ls_job_str ) );
    } else {
        job_s = (struct ls_job_sched_str *)data;
        strcpy ( ptr->job_name,     job_s->js_name     );
        strcpy ( ptr->job_source,   job_s->js_source   );
        strcpy ( ptr->job_location, ls->ls_hosts[0].host_name );
        ptr->job_user_id   = job_s->js_user_id;
        ptr->job_euser_id  = job_s->js_euser_id;
        ptr->job_group_id  = job_s->js_group_id;
        ptr->job_egroup_id = job_s->js_egroup_id;
        ptr->job_pid       = pid;
        ptr->job_priority  = 0;
```

```
        ptr->job_start      = time ( (time_t *)0 );
    }

/*
 *  Increment the number of jobs.
 */

    ls->ls_num_jobs++;
    ls->ls_last_job_upd = time ( (time_t *)0 );

/*
 *  Generate a status message.
 */

    sprintf ( string, "Job scheduled on host %s - Process ID is %d",
              ptr->job_location, ptr->job_pid );
    send_status_info ( ls, string );

/*
 *  If the job originated on a different host, send a status message back
 *  to allow that job to be properly tracked.
 */

    if ( strcmp ( ptr->job_source, ls->ls_hosts[0].host_name ) )
        if ( net_send_datagram ( ptr->job_source, ls->ls_port, (char *)ptr,
                sizeof ( struct ls_job_str ) ) == -1 )
            err_log ( f, "Could not send job status back to source", 0, 0 );

/*
 *  Always return a zero error code.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( best_host )
 *
 *  This function scans for host with lowest load.
 *****************************************************************************/


int best_host ( ls )

    struct ls_shm_str   *ls;
{
    register int    index = 0,
                    best  = 0,
                    delta,
                    i;

    D(printf("\nBEST_HOST\n"));
/*
 *  Loop through each remote host entry to find the host with the lowest load. Note that
 *  the "lowest load" is the difference between the hosts low value and its load. This
 *  calculation is necessary, as individual hosts may change their low value.
 */

    for ( i = 1; i < ls->ls_num_hosts; i++ )
        if ( ( delta = ls->ls_hosts[i].host_low_value -
                       ls->ls_hosts[i].host_current_load ) > best ) {
            index = i;
            best  = delta;
        }

/*
 *  Return the index of the host with the lowest load. This will be zero if no
 *  applicable host was found.
 */

    return ( index );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( child_termination )
 *
 *  This function handles termination of child processes.
 *****************************************************************************/


void child_termination ( sig, code, scp, addr )

    int                 sig,
                        code;

    struct  sigcontext  *scp;

    char                *addr;
{
    register int    pid;

    D(printf("\nCHILD_TERMINATION\n"));
/*
 *  Do a non-blocking wait for any terminated processes. Call (remove_job) to
 *  remove jobs from job table.
 */

    while ( ( pid = wait3 ( (union wait *)0, WNOHANG, (struct rusage *)0 ) ) > 0 ) {
        D(printf ( "Process %d died\n", pid ));
        remove_job ( ls, pid );
    }

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cleanup )
 *
 *  This function is responsible for cleaning up any resources created by this
 *  process. Note that even though this process does not initialize any resources,
 *  this mechanism is left in place for future growth and to be consistent with
 *  other processes.
 *****************************************************************************/

int cleanup ( )
{
    static char f[] = "cleanup";

    D(printf("\nCLEANUP\n"));
/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( process_job )
 *
 *  This function attempts to schedule or distribute a job.
 *****************************************************************************/

int process_job ( ls, job, length )

    struct ls_shm_str       *ls;

    struct ls_job_sched_str *job;

    int                     length;
{
    register int            best,
                            local;

    static char             f[] = "process_job";

    struct ls_host_str      *host;

    int                     x;

    extern void             child_termination ( ),
                            shutdown          ( ),
                            terminate         ( );

    D(printf("\nPROCESS_JOB\n"));
/*
 *  Set some pointers and flags to make the following comparisons easier to
 *  code.
 */

    host  = (struct ls_host_str *)&ls->ls_hosts[0];
    local = ! strcmp ( host->host_name, job->js_source );

/*
 *  Determine if the job should be run on the local host. This is the case if
 *  any of the following is true:
 *
 *      o   The force local flag was set in the job structure (this might be set by
 *          the user or set because the job could not be scheduled remotely.
 *      o   The job was initiated locally and the load is less than the high value.
 *      o   The job was initiated remotely and the load is less than the low value.
 *      o   The job was initiated locally and there are no hosts to distribute
 *          the job to.
 *      o   The job is already local and the maximum number of schedule trys
 *          has been reached (done now to prevent the following code from send-
 *          ing a schedule request from local host to local host.
 *      o   The job was submitted locally and there is no host with a low enough
 *          load.
 *
 *  Note that local requests are distributed if the load is greater than the high
 *  mark. Remote requests are only honored if the load is less than the low mark.
 */

    x = 0;
    if ( ( ++x &&  job->js_force_local                                      ) ||
         ( ++x &&  local && host->host_current_load <= host->host_high_value ) ||
         ( ++x && !local && host->host_current_load <= host->host_low_value  ) ||
         ( ++x &&  local && ls->ls_num_hosts == 1                           ) ||
         ( ++x &&  local && job->js_trys + 1 > MAX_NUM_TRYS                 ) ||
         ( ++x &&  local && ( best = best_host ( ls ) ) == 0                )    ) {

/*
 *      Conditions are right, so attempt to schedule the job. Note that at this
 *      point, no more attempts are made to schedule the job. This may be added later.
 */

        D(printf ( "X Flag is %d\n", x ));
        schedule_local_job ( ls, job, length );

/*
 *  Otherwise, attempt to schedule remotely. First set the best host index to
 *  zero (indicating a fallout to the source host).
 */

    } else {
        D(printf ( "X Flag is %d\n", x ));
        best = 0;

/*
 *      Increment the number of job schedule attempts. If equal to the maximum
 *      number of trys, set the force local flag to cause the job to return to
 *      the source host (note that this will not occur if already on the source
 *      host.
 */

        job->js_trys++;
        if ( job->js_trys > MAX_NUM_TRYS )
            job->js_force_local = 1;

/*
 *      Max number of tries not exceeded, so scan the host table to find the
 *      host with the lowest load. If this function returns zero value, then
 *      no host has a low enough load to accept the job, so set force local
 *      flag to return the job to the source host.
 */

        else if ( ( best = best_host ( ls ) ) == 0 )
            job->js_force_local = 1;

/*
 *      At this point, the job is ready to be sent to the host with the lowest
 *      load (best > 0 ) or back to the source host (best = 0). Send a datagram
 *      to the appropriate host.
 */

        if ( net_send_datagram ( ls->ls_hosts[best].host_name,
                                 ls->ls_port, (char *)job, length ) == -1 )
            err_log ( f, "Could not send job to remote host", 0, 0 );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( remove_job )
 *
 *  This function removes a job.
 *****************************************************************************/

int remove_job ( ls, pid )

    struct ls_shm_str   *ls;

    int                 pid;
{
    register int                i;

    static char                 f[] = "remove_job";

    struct ls_job_str           *ptr;

    struct ls_job_kill_resp_str dead_job;

    char                        string[100];

    D(printf("\nREMOVE_JOB\n"));
/*
 *  Scan job table for the terminated job.
 */

    for ( i = 0; i < MAX_JOBS; i++ )
        if ( ls->ls_jobs[i].job_pid == pid )
            break;

/*
 *  Output an error if the pid did not match any in table.
 */

    if ( i == MAX_JOBS ) {
        err_log ( f, "Terminated job not found in job table", 0, 0 );
        return ( -1 );
    }

/*
 *  Save a pointer to the matched job.
 */

    ptr = &ls->ls_jobs[i];

/*
 *  Remove job from job table.
 */

    ls->ls_num_jobs--;
    ls->ls_last_job_upd = time ( (time_t *)0 );
    ls->ls_jobs[i].job_pid = -1;

/*
 *  Generate a status message.
 */

    sprintf ( string, "Job removed on host %s - Process ID is %d",
              ls->ls_hosts[0].host_name, pid );
    send_status_info ( ls, string );

/*
 *  If the source of the job was a different host, send a message to indicate
 *  termination of the job.
 */

    if ( strcmp ( ptr->job_source, ls->ls_hosts[0].host_name ) ) {
        dead_job.jk_type   = TYPE_DEAD_RESP;
        dead_job.jk_pid    = pid;
        dead_job.jk_status = 0;
        if ( net_send_datagram ( ptr->job_source, ls->ls_port, (char *)&dead_job,
                sizeof ( struct ls_job_kill_resp_str ) ) == -1 )
            err_log ( f, "Could not send job termination back to source", 0, 0 );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( schedule_local_job )
 *
 *  This function scheduless the local job.
 *****************************************************************************/


int schedule_local_job ( ls, job, length )

    struct ls_shm_str       *ls;

    struct ls_job_sched_str *job;

    int                     length;
{
    register int    i,
                    pid,
                    len;

    static char     f[] = "schedule_local_job";

    char            string[MAX_STRING + 1],
                    *argv [100],
                    *p;

    D(printf("\nSCHEDULE_LOCAL_JOB\n"));
/*
 *  Set up the actual filename and the display.
 */

    if ( job->js_needs_xterm )
        sprintf ( string, "exec xterm -display %s:0.0 -e %s",
                          job->js_source, job->js_name );
    else
        sprintf ( string, "exec %s -display %s:0.0",
                          job->js_name, job->js_source );

/*
 *  Set up the basic exec strings.
 */

    argv[0] = "/bin/sh";
    argv[1] = "-c";
    argv[2] = string;
    argv[3] = 0;

/*
 *  Build the environment string. Enhance later to only do when necessary.
 */

    i = 0;
    length -= sizeof ( struct ls_job_sched_str );
    p = (char *)job + sizeof ( struct ls_job_sched_str );

    while ( length ) {
        envp[i++] = p;
        len = strlen ( p ) + 1;
        p      += len;
        length -= len;
    }

/*
 *  Fork and exec the program.
 */

    if ( ( pid = fork ( ) ) == 0 ) {
        execve ( argv[0], argv, envp );
        err_log ( f, "Could not exec program", -1, errno );
        exit ( 1 );
    } else if ( pid == -1 )
        err_log ( f, "Could not fork program", -1, errno );
    else
        D(printf ( "program spawned - pid is %d\n", pid ));

/*
 *  Add the job to the job table in shared memory.
 */

    add_job ( ls, 0, (struct ls_message_str *)job, pid );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( set_priority_on_job )
 *
 *  This function changes the priority of a job.
 *****************************************************************************/

int set_priority_on_job ( ls, job_pri )

    struct ls_shm_str       *ls;

    struct ls_job_pri_str   *job_pri;
{
    static char     f[] = "set_priority_on_job";

    char            string[MAX_STRING + 1];

    extern void     child_termination ( );

    D(printf("\nSET_PRIORITY_ON_JOB\n"));
/*
 *  Determine if the job is on the current host. If so, set the priority.
 */

    if ( strcmp ( job_pri->jp_location, ls->ls_hosts[0].host_name ) == 0 ) {
        sprintf ( string, "renice %d -p %d &", job_pri->jp_priority, job_pri->jp_pid );
        signal ( SIGCHLD, SIG_IGN );
        system ( string );
        signal ( SIGCHLD, child_termination );

/*
 *      Call the (update_job) to actually update shared memory with the new priority.
 */

        update_job ( ls, job_pri->jp_pid, job_pri->jp_priority );

/*
 *  Otherwise, the location of the job is not the current host, send a message to the
 *  correct host to terminate the job.
 */

    } else
        if ( net_send_datagram ( job_pri->jp_location, ls->ls_port, (char *)job_pri,
                sizeof ( struct ls_job_pri_str ) ) == -1 )
            err_log ( f, "Could not send job priority change request", 0, 0 );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( terminate_job )
 *
 *  This function terminates a job.
 *****************************************************************************/

int terminate_job ( ls, job_kill )

    struct ls_shm_str       *ls;

    struct ls_job_kill_str  *job_kill;
{
    static char     f[] = "terminate_job";

    D(printf("\nTERMINATE_JOB\n"));
/*
 *  Determine if the job is on the current host. If so, attempt to kill it.
 */

    if ( strcmp ( job_kill->jk_location, ls->ls_hosts[0].host_name ) == 0 )
        kill ( job_kill->jk_pid, SIGTERM );

/*
 *  Otherwise, the location of the job is not the current host, send a message to the
 *  correct host to terminate the job.
 */

    else
        if ( net_send_datagram ( job_kill->jk_location, ls->ls_port, (char *)job_kill,
                sizeof ( struct ls_job_kill_str ) ) == -1 )
            err_log ( f, "Could not send job termination request", 0, 0 );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*************************************************************************************
 * MODULE NAME AND FUNCTION: ( update_job )
 *
 *  This function changes the priority of a job.
 *************************************************************************************/

int update_job ( ls, pid, priority )

    struct ls_shm_str   *ls;

    int                 pid,
                        priority;
{
    register int                i;

    static char                 f[] = "update_job";

    struct ls_job_str           *ptr;

    struct ls_job_pri_resp_str  job_pri;

    char                        string[100];

    D(printf("\nUPDATE_JOB\n"));
/*
 *  Search for the entry in the job table matching the pid of the priority request.
 */

    for ( i = 0; i < MAX_JOBS; i++ )
        if ( ls->ls_jobs[i].job_pid == pid )
            break;

/*
 *  Output an error if no match was found.
 */

    if ( i == MAX_JOBS ) {
        err_log ( f, "Job to be updated not found in job table", 0, 0 );
        return ( -1 );
    }

/*
 *  Update the priority and the time of last job update.
 */

    ptr = &ls->ls_jobs[i];
    ptr->job_priority = priority;

    ls->ls_last_job_upd = time ( (time_t *)0 );

/*
 *  Send a status message.
 */

    sprintf ( string, "Job priority updated on host %s - Process ID is %d",
              ls->ls_hosts[0].host_name, pid );
    send_status_info ( ls, string );

/*
 *  If the request originated on a different host, send a status message back
 *  to allow that job to be properly updated.
 */

    if ( strcmp ( ptr->job_source, ls->ls_hosts[0].host_name ) ) {
        job_pri.jp_type     = TYPE_PRIORITY_RESP;
        job_pri.jp_pid      = pid;
        job_pri.jp_priority = priority;
        if ( net_send_datagram ( ptr->job_source, ls->ls_port, (char *)&job_pri,
                sizeof ( struct ls_job_pri_resp_str ) ) == -1 )
            err_log ( f, "Could not send job priority status back to source", 0, 0 );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

ORIGINAL PAGE IS OF POOR QUALITY

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( ls_status )
 *
 *  This program is a daemon process which is responsible for processing all in-
 *  coming network datagrams. This process waits on the load sharing port and
 *  when a datagram arrives, outputs the datagram on the load sharing message
 *  queue. This process is kept very simple to insure that it can effectively
 *  process all incoming datagrams.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      ls_status
 *      attach_to_has
 *      calculate_load
 *      check_status_queue
 *      cleanup
 *      get_raw_stats
 *      host_index
 *      remove_stale_hosts
 *      send_status_data
 *
 *
 * COMMAND LINE EXECUTION:
 *
 *  ls_status
 *
 *
 * REQUIRED EXTERNAL RESOURCES:
 *
 *  o   The shared memory segment used to contain all load sharing data.
 *
 *  o   The TCP/IP port used for transmission of all datagrams. The port
 *      number is retrieved from shared memory.
 *
 *  o   The message queue used to queue up all messages and requests for the
 *      load sharing system.
 *
 *  o   The semaphore used to control access to shared memory.
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <has.h>
#include <configure.h>
#include <structs.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <ls.h>
#include <ls_status.h>


struct resource_str resources;

char                process[] = "ls_status";

extern void         shutdown  ( ),
                    terminate ( );

extern int          errno;


main ( argc, argv )

    int     argc;

    char    **argv;
{
    register int        advert_int,
                        heartbeat_int,
                        check_host_int;

    static char         f[] = "ls_status";

    struct ls_shm_str   *ls;

    struct has          *ls_has;

    D(printf("\nLS_STATUS\n"));
/*
 *  Initialize signals.
 */

    init_signals ( shutdown, terminate );

/*
 *  Attach to has shared memory.
 */

    if ( attach_to_has ( &ls_has ) == -1 ) {
        cleanup ( );
        exit ( 1 );
    }

/*
 *  Attach to shared memory.
 */

    if ( attach_shared_memory ( &ls ) == -1 ) {
        cleanup ( );
        exit ( 1 );
    }

/*
 *  Initialize intervals. Note that there is no reason to lock semaphore for shared
 *  memory access for read-only interval values.
 */

    advert_int     = ls->ls_advert_int;
    heartbeat_int  = ls->ls_heartbeat_int;
    check_host_int = ls->ls_check_host_int;

/*
 *  Begin a loop which continues forever and processes outgoing and incoming status
 *  messages.
 */

    while ( 1 ) {
        sleep ( ls->ls_status_int );
        sem_lock ( ls->ls_sem_id );
```

```
        get_raw_stats ( ls, ls_has );
        if ( !advert_int || !heartbeat_int ) {
            advert_int = ls->ls_advert_int;
            calculate_load ( ls );
            if ( send_status_data ( ls ) || !heartbeat_int ) {
                if ( net_send_datagram ( 0, ls->ls_port,
                     (char *)&ls->ls_hosts[0], sizeof ( struct ls_host_str ) ) == -1 ) {
                    cleanup ( );
                    exit ( 1 );
                }
                heartbeat_int = ls->ls_heartbeat_int;
            }
        }

/*
 *      Check the status queue to determine if any status buffers have been received fro
m
 *      other hosts.
 */

        if ( check_status_queue ( ls ) == -1 ) {
            cleanup ( );
            exit ( 1 );
        }

/*
 *      If the interval has been reached, scan through the host table to determine if
 *      any hosts have not responded in the defined timeout period. Such hosts will be
 *      removed from the table.
 */

        if ( !check_host_int ) {
            remove_stale_hosts ( ls );
            check_host_int = ls->ls_check_host_int;
        }

/*
 *  Unlock the semaphore.
 */

        sem_unlock ( ls->ls_sem_id );

/*
 *      Decrement each interval by the amount waited for the main interval (the status
 *      interval, which is always the shortest).
 */

        advert_int     -= ls->ls_status_int;
        heartbeat_int  -= ls->ls_status_int;
        check_host_int -= ls->ls_status_int;
    }
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( attach_to_has )
 *
 *  This function attaches to (has) shared memory.
 *****************************************************************************/

int attach_to_has ( shm_addr )

    struct has  **shm_addr;
{
    static char f[] = "attach_to_has";

    int         shm_id;

    extern char *shmat ( );

    D(printf("\nATTACH_TO_HAS\n"));
/*
 *  Retrieve the shared memory id associated with the has shared memory.
 */

    if ( ( shm_id = shmget ( SHARED_MEM_KEY, 0, 0 ) ) == -1 )
        return ( err_log ( f, "Could not get has shared memory id", -1, errno ) );

/*
 *  Attach to the shared memory and set address.
 */

    if ( (int)( *shm_addr = (struct has *)shmat ( shm_id, 0, 0 ) ) == -1 )
        return ( err_log ( f, "Could not attach to has shared memory", -1, errno ) );

/*
 *  Return the shared memory ID.
 */

    return ( shm_id );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( calculate_load )
 *
 *  This function calculates the current load.
 *****************************************************************************/

int calculate_load ( ls )

    struct ls_shm_str   *ls;
{
    register int    avg_users  = 0,
                    avg_procs  = 0,
                    avg_cpu    = 0,
                    avg_memory = 0,
                    i,
                    load;

    D(printf("\nCALCULATE_LOAD\n"));
/*
 *  Accumulate average statistics. A -1 is when no status has yet been recorded. Seems
 *  to be a waste.
 */

    for ( i = 0; i < PARM_ADVERT_INT; i++ )
        if ( ls->ls_raw_stats[i].rs_cpu != -1 ) {
            avg_users  += ls->ls_raw_stats[i].rs_users;
            avg_procs  += ls->ls_raw_stats[i].rs_procs;
            avg_cpu    += ls->ls_raw_stats[i].rs_cpu;
            avg_memory += ls->ls_raw_stats[i].rs_memory;
        }

/*
 *  Calculate the load. Right now this is just the average used CPU. Ultimately, this
 *  will take into account memory, risc, cpu, etc.  Save the load value into the
 *  current host structure.
 */

    ls->ls_hosts[0].host_current_load = 100 - avg_cpu / ( PARM_ADVERT_INT * 10 );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( check_status_queue )
 *
 *  This function determines if any incomine status messages are pending.
 *****************************************************************************/

int check_status_queue ( ls )

    struct ls_shm_str   *ls;
{
    register int            i,
                            length;

    static char             f[] = "check_status_queue";

    struct ls_message_str   message;

    struct ls_host_str      *ptr;

    D(printf("\nCHECK_STATUS_QUEUE\n"));
/*
 *  Begin a loop which will process each pending message. This loop continues until
 *  no more messages are pending.
 */

    errno = 0;

    while ( errno != ENOMSG ) {
        if ( ( length = msgrcv ( ls->ls_mq_id, (struct msgbuf *)&message, MAX_MESSAGE,
                                 TYPE_STATUS, IPC_NOWAIT ) ) == -1 ) {
            if (  errno != ENOMSG )
                return ( err_log ( f, "Could not read status from queue", -1, errno ) );

/*
 *      A message has arrived. Save a pointer to the data area of the message. The
 *      message will contain a host structure.
 */

        } else {
            ptr = (struct ls_host_str *)&message;

/*
 *          Determine whether or not this message is from a new or old host. If old,
 *          update the current entry in the host table. If new, increment the number
 *          of hosts and add the new host at the end of the list. Note that an error
 *          is logged and the message discarded if the host cannot be added due to the
 *          size of the table.
 */

            i = host_index ( ls, ptr->host_name );

            if ( i != -1 )
                memcpy ( (char *)&ls->ls_hosts[i], (char *)ptr,
                         sizeof ( struct ls_host_str ) );
            else if ( ls->ls_num_hosts + 1 >= MAX_HOSTS ) {
                err_log ( f, "Host table full, can't add new host", 0, 0 );
                send_status_info ( ls, "Host table full, can't add new host" );
            }else {
                i = ls->ls_num_hosts++;
                memcpy ( (char *)&ls->ls_hosts[i], (char *)ptr,
                         sizeof ( struct ls_host_str ) );
                ls->ls_last_host_upd = time ( (time_t *)0 );
            }
```

```
/*
 *  Update the timestamp of the arrived datagram.
 */

            ls->ls_hosts[i].host_timestamp = time ( (time_t *)0 );
        }
    }

/*
 *  Successfull return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cleanup )
 *
 *  This function is responsible for cleaning up any resources created by this
 *  process. Note that even though this process does not initialize any resources,
 *  this mechanism is left in place for future growth and to be consistent with
 *  other processes.
 *****************************************************************************/

int cleanup ( )
{
    static char f[] = "cleanup";

    D(printf("\nCLEANUP\n"));
/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( get_raw_stats )
 *
 *  This function retrieves raw statistics from (has) shared memory.
 *****************************************************************************/

int get_raw_stats ( ls, has )

    struct ls_shm_str   *ls;

    struct has          *has;
{
    register int    ptr;

    D(printf("\nGET_RAW_STATS\n"));
/*
 *  Save array pointer to make access easier and faster.
 */

    ptr = ++(ls->ls_stats_ptr);
    if ( ptr == PARM_ADVERT_INT )
        ptr = ls->ls_stats_ptr = 0;

/*
 *  Save data into the appropriate array cell.
 */

    ls->ls_raw_stats[ptr].rs_users  = has->procstats.users;
    ls->ls_raw_stats[ptr].rs_procs  = has->procstats.total_p;
    ls->ls_raw_stats[ptr].rs_cpu    = has->cpu_percent[0] ;
    ls->ls_raw_stats[ptr].rs_memory = has->total_mem_used;

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( host_index )
 *
 *  This function converts a host into an index into the host table.
 *****************************************************************************/

int host_index ( ls, hostname )

    struct ls_shm_str   *ls;

    char                *hostname;
{
    register int    i;

    D(printf("\nHOST_INDEX\n"));
/*
 *  Can for the host which matches the passed host name.
 */

    for ( i = 0; i < ls->ls_num_hosts; i++ )
        if ( strcmp ( hostname, ls->ls_hosts[i].host_name ) == 0 )
            return ( i );

/*
 *  Return error (not found) at this point.
 */

    return ( -1 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( remove_stale_hosts )
 *
 *  This function removes any hosts no longer active in load sharing.
 *****************************************************************************/

int remove_stale_hosts ( ls )

    struct ls_shm_str   *ls;
{
    static char     f[] = "remove_stale_hosts";

    char            s[MAX_STRING + 1];

    register int    i, j,
                    secs;

    D(printf("\nREMOVE_STALE_HOSTS\n"));
/*
 *  Save the current time for comparison purposes.
 */

    secs = time ( (time_t *)0 );

/*
 *  Loop through and check if the last message received from a host is more than the
 *  timeout value. Note that no check is made to ensure that the current host is not
 *  deleted (would only be due to a bug).
 */

    for ( i = 0; i < ls->ls_num_hosts; i++ )
        if ( secs - ls->ls_hosts[i].host_timestamp > ls->ls_host_timeout ) {

/*
 *          If a stale host was found, decrement the number of active hosts. Next,
 *          if the host was not the last in the list, shift each host entry down one.
 */

            sprintf ( s, "No status from %s, host removed", ls->ls_hosts[i].host_name );
            err_log ( f, s, 0, 0 );
            send_status_info ( ls, s );

            ls->ls_num_hosts--;
            ls->ls_last_host_upd = time ( (time_t *)0 );
            for ( j = i; j < ls->ls_num_hosts; j++ )
                memcpy ( (char *)&ls->ls_hosts[j], (char *)&ls->ls_hosts[j + 1],
                         sizeof ( struct ls_host_str ) );
        }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( send_status_data )
 *
 *  This function determines if it is necessary to send data.
 *****************************************************************************/

int send_status_data ( ls )

    struct ls_shm_str   *ls;
{
    static int      old_load = -1;

    register int    load,
                    low_value,
                    ret_value;

    D(printf("\nSEND_STATUS_DATA\n"));
/*
 *  Save the load and low_value as they are used frequently.
 */

    low_value = ls->ls_hosts[0].host_low_value;
    load      = ls->ls_hosts[0].host_current_load;

/*
 *  If the load is less than the low value (allow introduction of new jobs) and the
 *  load has changed from the previous, set the return value to true. Also, if the load
 *  is greater than the low value now, but has previously less than the low value, set
 *  the return value to true. Note that this covers both situations where the load
 *  crosses the low line.
 */

    if ( load <= low_value && load     != old_load  ||
         load >  low_value && old_load <= low_value    )
        ret_value = 1;
    else
        ret_value = 0;

/*
 *  Save the current load value.
 */

    old_load = load;

/*
 *  Return flag.
 */

    return ( ret_value );
}
```

```
/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( ls_uilib.c )
 *
 *  This library contains user interface functions which simplify interfacing to
 *  Motif.
 *
 *
 * INTERNAL FUNCTIONS:
 *
 *      bad_syntax
 *      create_cascade
 *      create_command
 *      create_form
 *      create_label
 *      create_text
 *      create_toggle
 *      display_message
 *
 *
 * ORIGINAL PROGRAMMER:
 *
 *  Mark D. Collier -   Software Engineering Section
 *                      Data Systems Department
 *                      Automation and Data Systems Division
 *                      Southwest Research Institute
 *****************************************************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/MwmUtil.h>
#include <Xm/CascadeB.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include <ls.h>


/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( create_cascade )
 *
 *  This function is called to create a MOTIF cascade widget.
 *****************************************************************************
*/

Widget create_cascade ( instance, parent, menu, label )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

            *label;             /* The string which this command widget will display
                                 */

    Widget  parent,             /* The parent widget to which the command widget will
                                 * be attached.
                                 */

            menu;               /* Menu which will be activated when the cascade is
                                 * selected.
                                 */

{
/*
 *  Define the array which will contain all arguments required to create the command
 *  widget.
 */

    Widget          widget;     /* Pointer to the created widget.
                                 */

    Arg             args[ 1 ];  /* Argument list for cascade widget.
                                 */

    register int    count = 0;  /* Counts the number of arguments initialized.
                                 */

/*
 *  attach it to the parent, and initialize all arguments.
 */

    XtSetArg ( args[ count ], XmNsubMenuId, menu ); count++;

/*
 *  Create and manage the cascade widget. Return the widget pointer to calling function.
 */

    XtManageChild ( widget = XmCreateCascadeButton ( parent, label, args, count ) );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
 * MODULE NAME AND FUNCTION ( create_command )
 *
 *  This function is called to create a command widget.
 *****************************************************************************
/

Widget create_command ( instance, parent, label, callback )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

            *label;             /* The string which this command widget will display.
                                 */

    Widget  parent;             /* The parent widget to which the command widget will
                                 * be attached.
```

```
                                */

    XtCallbackList  callback;       /* Specifies an array containing the list of func-
                                     * tions called upon command callback.  It may be
                                     * NULL if no functions are present.
                                     */
{
    Widget          widget;         /* Pointer to the created widget.
                                     */

    XmString        string;         /* Compound string to which the label is converted.
                                     */

    Arg             args[ 2 ];      /* Argument list used to initialize widget resources
                                     */

    register int    count = 0;      /* Incremented each time a argument is initialized
                                     * in the (args) array.  When the widget is created,
                                     * this value which indicate the number of arguments
                                     */

/*
 *  Convert the label to a compound string and save in the argument list.
 */

    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[ count ], XmNlabelType,  XmSTRING ); count++;
    XtSetArg ( args[ count ], XmNlabelString, string  ); count++;

/*
 *  Create and manage the widget. Free the memory allocated for the compound string.
 */

    XtManageChild ( widget = XmCreatePushButton ( parent, instance, args, count ) );
    XmStringFree ( string );

/*
 *  If the command has a callback, add it to the widget.
 */

    if ( callback )
        XtAddCallbacks ( widget, XmNactivateCallback, callback );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( create_form )
 *
 *  This function is called to create a MOTIF form widget.
 *****************************************************************************
*/

Widget create_form ( instance, parent )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance;          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

    Widget  parent;             /* The parent widget to which the form widget will
                                 * be attached.
                                 */
{
    Widget      widget;         /* Pointer to the created widget.
                                 */

/*
 *  Create and manage the form widget. Return the widget pointer to the calling function.
 */

    XtManageChild ( widget = XmCreateForm ( parent, instance, NULL, 0 ) );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
 * MODULE NAME AND FUNCTION ( create_label )
 *
 *  This function is called to create a MOTIF label widget.
 *****************************************************************************
/

Widget create_label ( instance, parent, label )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

            *label;             /* The string which this label widget will display.
                                 */

    Widget  parent;             /* The parent widget to which the label widget will
                                 * be attached.
                                 */
{
    Arg args[ 2 ];              /* Argument list used to initialize the widget
                                 * resources.
                                 */

    Widget      widget;         /* Pointer to the created widget.
                                 */

    XmString    string;         /* Points to the compound string created for the
                                 * label.
                                 */

    register int    count = 0;  /* Counter set to the number of arguments initialized
```

```
                                */

/*
 *  Initialize a compound string and set in the resource list.
 *  lists.
 */

    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[ count ], XmNlabelType,   XmSTRING ); count++;
    XtSetArg ( args[ count ], XmNlabelString, string   ); count++;

/*
 *  Create and manage the widget. Free the space allocated for the compound string.
 *  Return the widget pointer to the calling function.
 */

    XtManageChild ( widget = XmCreateLabel ( parent, instance, args, count ) );
    XmStringFree ( string );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( create_text )
 *
 *  This function is called to create a MOTIF text widget.
 *****************************************************************************
*/

Widget create_text ( instance, parent, text, scrolled, mode, edit_flag )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

            *text;              /* The ascii text which will be displayed in the
                                 * text widget.
                                 */

    Widget  parent;             /* The parent widget to which the text widget will
                                 * be attached.
                                 */

    int     mode,               /* Indicates whether the widget will be single or
                                 * multiple lines:
                                 *
                                 *   XmSINGLE_LINE_EDIT
                                 *   XmMULTI_LINE_EDIT
                                 */

            scrolled,           /* Indicates whether or not the data can be scrolled
                                 */

            edit_flag;          /* Indicates whether or not the widget can be edited
                                */

{
    Widget          widget;     /* Pointer to the created widget.
                                 */

    Arg             args[ 3 ];  /* Argument list used to initialize the widget
                                 * resources.
                                 */

    register int    count = 0;  /* Used to count the number of arguments initialized.
                                 */

/*
 *  Initialize the widget text (not a compound string), the line size mode, and the
 *  edit mode.
 */

    XtSetArg ( args[ count ], XmNvalue,    text      ); count++;
    XtSetArg ( args[ count ], XmNeditMode, mode      ); count++;
    XtSetArg ( args[ count ], XmNeditable, edit_flag ); count++;

/*
 *  Based on the (scrolled) flag, create the appropriate type of widget. Next manage
 *  the widget. Note that the instance name of a scrolled text widget is "instanceSW".
 */

    if ( scrolled )
        widget = XmCreateScrolledText ( parent, instance, args, count );
    else
        widget = XmCreateText         ( parent, instance, args, count );
    XtManageChild ( widget );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
 * MODULE NAME AND FUNCTION ( create_toggle )
 *
 *  This function is called to create a toggle button widget.
 *****************************************************************************
/

Widget create_toggle ( instance, parent, label )

                                /* This function returns the return value of the
                                 * XtCreateManagedWidget function call.  This will
                                 * be a pointer to a widget.
                                 */

    char    *instance,          /* The instance name of the widget.  It uniquely
                                 * defines the widget.
                                 */

            *label;             /* The string which this label widget will display.
                                 */

    Widget  parent;             /* The parent widget to which the label widget will
                                 * be attached.
                                 */

{
```

```
    Arg args[ 2 ];                  /* Argument list used to initialize the widget
                                     * resources.
                                     */

    Widget      widget;             /* Pointer to the created widget.
                                     */

    XmString    string;             /* Points to the compound string created for the
                                     * label.
                                     */

    register int    count = 0;      /* Counter set to the number of arguments initialize
d.
                                     */

/*
 *  Convert the label to a compound string and initialize in the argument list.
 */

    string = XmStringLtoRCreate ( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[ count ], XmNlabelType,   XmSTRING ); count++;
    XtSetArg ( args[ count ], XmNlabelString, string );   count++;

/*
 *  Create and manage the widget. Free the space allocated for the compound string.
 *  Return the widget pointer to the calling function.
 */

    XtManageChild ( widget = XmCreateToggleButton ( parent, instance, args, count ) );
    XmStringFree ( string );

/*
 *  Return widget.
 */

    return ( widget );
}


/*****************************************************************************
 * MODULE NAME AND FUNCTION ( display_message )
 *
 *  This function displays different types of popups for different message types. It dis
-
 *  plays a modal popup which when acknowledged, is automatically removed. This function
 *  also calls (message). Note that all user interface clients should use this function
 *  to present messages.
 *****************************************************************************
*/

int display_message ( top, type, message_text )
                                    /* Function returns the return value of message
                                     * call.
                                     */

    Widget  top;                    /* Main window of the application. Used to attach
                                     * the shell widget.
                                     */

    int     type;                   /* Type of the message. Used to determine the type
                                     * of popup displayed:
                                     *
                                     *  MSG_ERROR   1
                                     *  MSG_INFO    2
                                     *  MSG_WARN    3
                                     */

    char    *message_text;          /* Message text to actually display.
                                     */
{
    Arg             args[ 1 ];      /* Argument list used to initialize the widget
                                     * resources.
                                     */

    static Widget   widget;         /* Pointer to the created widget.
                                     */

    XmString        string;         /* Points to the compound string created for the
                                     * label.
                                     */

    register int    count = 0;      /* Counts the number of arguments.
                                     */

/*
 *  If a popup was already defined, destroy it (it will have been unmanaged, but
 *  will still exist.
 */

    if ( widget )
        XtDestroyWidget ( widget );

/*
 *  Initialize the string to be displayed in the popup.
 */

    string = XmStringLtoRCreate ( message_text, XmSTRING_DEFAULT_CHARSET );
    XtSetArg ( args[ count ], XmNmessageString, string ); count++;

/*
 *  Based on the message type, create the appropriate popup type.
 */

    switch ( type ) {

    case    MSG_INFO:
        widget = XmCreateInformationDialog ( top, "", args, count );
        break;
    case    MSG_ERROR:
        widget = XmCreateErrorDialog       ( top, "", args, count );
        break;
    case    MSG_WARN:
        widget = XmCreateWarningDialog     ( top, "", args, count );
        break;
    default:
        break;
    }

/*
 *  Set the modal flag on the popup shell widget.
 */

    count = 0;
    XtSetArg ( args[ count ], XmNmwmInputMode, MWM_INPUT_APPLICATION_MODAL ); count++;
    XtSetValues ( XtParent ( widget ), args, count );

/*
 *  Manage the widget and Free the string used for the compound string.
 */
```

```
    XtManageChild ( widget );
    XmStringFree ( string );

/*
 *  Unmanage the CANCEL and HELP push buttons as they have no function.
 */

    XtUnmanageChild ( XmMessageBoxGetChild ( widget, XmDIALOG_CANCEL_BUTTON ) );
    XtUnmanageChild ( XmMessageBoxGetChild ( widget, XmDIALOG_HELP_BUTTON   ) );

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************************
*
 * MODULE NAME AND FUNCTION ( cbr_parm_remove )
 *
 *  This callback function removes the statis parameters popup window.
 *****************************************************************************************
/

XtCallbackProc cbr_parm_remove ( widget, closure, calldata )

    Widget  widget;                 /* Set to the widget which initiated this callback
                                     * function.
                                     */

    caddr_t closure,                /* Callback specific data.  This parameter is the
                                     * the main child widget of the popup.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client. This parameter is
                                     * not used by this function.
                                     */

{
    D(printf("\nCBR_PARM_REMOVE\n"));
/*
 *  Unmanage (unmap) the main child widget of the static parameters popup window.
 */

    XtUnmanageChild ( (Widget)closure );

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************************
 * MODULE NAME AND FUNCTION ( cbr_run_job )
 *
 *  This callback function initiates execution of a job.
 *****************************************************************************************/

XtCallbackProc cbr_run_job ( widget, closure, calldata )

    Widget  widget;                 /* Set to the widget which initiated this callback
                                     * function.
                                     */

    caddr_t closure,                /* Callback specific data.  This parameter indicates
                                     * which command button initiated this callback.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    char        *job;

    D(printf("\nCBR_RUN_JOB\n"));
/*
 *  If called from the command within the main window, manage the popup used to prompt
 *  for the job name to execute.
 */

    if ( (int)closure == 0 )
        XtManageChild ( run_popup );

/*
 *  Otherwise the callback initiated from the popup itself. If the OK or APPLY commands
 *  was selected, then the user wants to execute a job.
 */

    else
        if ( (int)closure == 1 || (int)closure == 2 ) {

/*
 *          Get the string specified by the user (the job name). If blank, generate a
 *          warning message.
 */

            job = XmTextGetString ( XmSelectionBoxGetChild ( run_popup, XmDIALOG_TEXT ) )
;
            if ( *job == 0 ) {
                XtManageChild ( run_popup );
                display_message ( top, MSG_WARN, "You must specify a job name" );

/*
 *          Otherwise, initialize the job structure with a flag which indicates whether
 *          or not an xterm should be used and call a function to schedule the job.
 */

            } else {
                job_buffer->js_needs_xterm = ( (int)closure == 1 ) ? 1 : 0;
                schedule_job ( ls, job_buffer, total, job );
                XtUnmanageChild ( run_popup );
            }

/*
 *      Otherwise the user canceled the job, so simply remove the popup.
```

```
 */

        } else
            XtUnmanageChild ( run_popup );

/*
 *  Normal return.
 */

    return;

}
```

```
/*****************************************************************************************
 * MODULE NAME AND FUNCTION ( cbr_set_priority )
 *
 *  This callback function allows the user to set the priority on a job.
 *****************************************************************************************/

XtCallbackProc cbr_set_priority ( widget, closure, calldata )

    Widget  widget;                 /* Set to the widget which initiated this callback
                                     * function.
                                     */

    caddr_t closure,                /* Callback specific data.  This parameter is not
                                     * used by this function.
                                     */
            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    register int            i,
                            pid;

    static int              index;

    Arg                     args[10];

    XmStringTable           strings;

    char                    string[200 + 1],
                            *p = "";

    int                     count,
                            priority;

    D(printf("\nCBR_SET_PRIORITY\n"));
/*
 *  If called from the main menu, prepare to verify the termination of the job. First,
 *  extract the current selection from the widget. If no selection was made, display
 *  a warning to the user and return.
 */

    if ( (int)closure == 0 ) {
        i = 0;
        XtSetArg ( args[i], XmNselectedItems,     &strings ); i++;
        XtSetArg ( args[i], XmNselectedItemCount, &count   ); i++;
        XtGetValues ( jobs, args, i );

        if ( count == 0 ) {
            display_message ( top, MSG_WARN, "No job selection made" );
            return;
        }

/*
 *      Convert the first entry in the strings array (the first selection) to a normal
 *      character string and parse out the first substring (the process ID).
 */

        XmStringGetLtoR ( *strings, DC, &p );
        pid = atoi ( p );

/*
 *      Match the process ID from the selection to the entry in the job table to
```

```
 *      which it corresponds.
 */

        for ( index = 0; index < MAX_JOBS; index++ )
            if ( ls->ls_jobs[index].job_pid == pid )
                break;

/*
 *      Display an error if no match was found. Display a warning if the user does
 *      not own the selected job.
 */

        if ( index == MAX_JOBS ) {
            display_message ( top, MSG_ERROR, "Job not found in job table - reselect" );
            return;
        }

        if ( ls->ls_jobs[index].job_user_id != getuid ( ) ) {
            display_message ( top, MSG_WARN, "You do not own that job" );
            return;
        }

/*
 *      Set the scale to the current priority value.
 */

        XmScaleSetValue ( pscale, ls->ls_jobs[index].job_priority );

/*
 *      All ready, display the popup to allow verification.
 */

        XtManageChild ( pri_popup );

/*
 *  Otherwise, the call was from the popup. If the user selected the OK button, call
 *  function to initiate the set priority request.
 */

    } else {
        if ( (int)closure == 1 ) {
            XmScaleGetValue ( pscale, &priority );
            set_priority ( ls, index, priority );
        }
        XtUnmanageChild ( pri_popup );
    }

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************************
 * MODULE NAME AND FUNCTION ( cbr_show_host )
 *
 *  This callback function creates a popup which contains information for a remote
 *  host.
 *****************************************************************************************/

XtCallbackProc cbr_show_host ( widget, closure, calldata )

    Widget  widget;                 /* Set to the widget which initiated this callback
                                     * function.
                                     */

    caddr_t closure,                /* Callback specific data.  This parameter is not
                                     * used by this function.
                                     */

            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    register int            i,
                            index;

    static XtCallbackRec    cb[] = {
        { (XtCallbackProc)NULL, (caddr_t)NULL },
        { (XtCallbackProc)NULL, (caddr_t)NULL }
    };

    Widget                  shell, m_main, mb_main, mp_file, w, frame, form,
                            lscale, lvscale, hvscale;

    Arg                     args[10];

    XmStringTable           strings;

    char                    string[MAX_STRING + 1],
                            *p = "";

    int                     count;

    XtCallbackProc          cbr_host_remove ( );

    D(printf("\nCBR_SHOW_HOST\n"));
/*
 *  Get the list of currently selected strings in the host list. If at least one
 *  item is currently selected (count), convert the selection from a compound string
 *  to a normal string (p).
 */

    i = 0;
    XtSetArg ( args[i], XmNselectedItems,     &strings ); i++;
    XtSetArg ( args[i], XmNselectedItemCount, &count   ); i++;
    XtGetValues ( hosts, args, i );

    if ( count )
        XmStringGetLtoR ( *strings, DC, &p );

/*
 *  Verify that it is valid to display a new popup. This function converts the selected
 *  string to an index into the shared memory host table.
 */
```

```
    if ( ( index = ok_to_add_host ( p ) ) == -1 )
        return;

/*
 *  Create the shell widget which is the parent of the window. Set the name of the
 *  window to identify the host selected.
 */

    sprintf ( string, "%s Parameters", p );
    shell = XmCreateDialogShell ( top, string, NULL, 0 );

/*
 *  Create the main window widget and the menu bar which will contain all main window
 *  commands.
 */

    XtManageChild ( m_main  = XmCreateMainWindow ( shell,    "", NULL, 0 ) );
    XtManageChild ( mb_main = XmCreateMenuBar    ( m_main, "", NULL, 0 ) );

/*
 *  Create pulldown menu, cascade button, and command widget for file-related
 *  commands.
 */

    mp_file = XmCreatePulldownMenu ( mb_main, "", NULL, 0 );
    create_cascade ( "", mb_main, mp_file, "File" );

    cb[0].callback = (XtCallbackProc)cbr_host_remove;
    cb[0].closure  = (caddr_t)index;
    create_command ( "", mp_file, "Remove", cb );

/*
 *  Create the help cascade and force it to appear on the right edge of the menu
 *  bar.
 */

    w = create_cascade ( "", mb_main, NULL, "Help" );
    XtSetArg ( args[0], XmNmenuHelpWidget, w );
    XtSetValues ( mb_main, args, 1 );

/*
 *  Create the frame and form which will contain the scale widgets for the current
 *  load, the low value, and the high value. The form is named to allow resources
 *  to by set for the child scale widgets (set to insensitive, since the user can
 *  not change them).
 */

    XtManageChild ( frame = XmCreateFrame ( m_main, "sframe", NULL, 0 ) );
    form = create_form ( "hostform", frame );

/*
 *  Create the separators which separate the scales.
 */

    XtManageChild ( XmCreateSeparator ( form, "sep0", NULL, 0 ) );
    XtManageChild ( XmCreateSeparator ( form, "sep1", NULL, 0 ) );

/*
 *  Create the labels which identify the three scales.
 */

    create_label ( "slabel0", form, "Local Load" );
    create_label ( "slabel1", form, "Low Mark"   );
    create_label ( "slabel2", form, "High Mark"  );

/*
 *  Create the scale used for the current load. Include two labels indicating the
 *  high and low limits of 100 and 0.
 */

    XtManageChild ( lscale  = XmCreateScale ( form, "scale0", NULL, 0 ) );

    create_label ( "", lscale, "  100" );
    create_label ( "", lscale, "    0" );

/*
 *  Create the scale used to allow the user to adjust the low value. Include the
 *  minimum and maximum values and add corresponding labels.
 */

    i = 0;
    XtSetArg ( args[i], XmNminimum, ls->ls_low_value_min ); i++;
    XtSetArg ( args[i], XmNmaximum, ls->ls_low_value_max ); i++;
    XtSetArg ( args[i], XmNvalue,   ls->ls_hosts[index].host_low_value ); i++;
    XtManageChild ( lvscale = XmCreateScale ( form, "scale1", args, i ) );
    create_label ( "", lvscale, sprintf ( string, "%5d", ls->ls_low_value_max  ) );
    create_label ( "", lvscale, sprintf ( string, "%5d", ls->ls_low_value_min  ) );

/*
 *  Create the scale used to allow the user to adjust the high value. Include the
 *  minimum and maximum values and add corresponding labels.
 */

    i = 0;
    XtSetArg ( args[i], XmNminimum, ls->ls_high_value_min ); i++;
    XtSetArg ( args[i], XmNmaximum, ls->ls_high_value_max ); i++;
    XtSetArg ( args[i], XmNvalue,   ls->ls_hosts[index].host_high_value ); i++;
    XtManageChild ( hvscale = XmCreateScale ( form, "scale2", args, i ) );
    create_label ( "", hvscale, sprintf ( string, "%5d", ls->ls_high_value_max ) );
    create_label ( "", hvscale, sprintf ( string, "%5d", ls->ls_high_value_min ) );

/*
 *  Update the table which controls all displayed host popup windows.
 */

    disp_hosts[index].dh_shell_widget = shell;
    disp_hosts[index].dh_scale_widget = lscale;

/*
 *  Realize all widgets.
 */

    XtRealizeWidget ( shell );

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( cbr_show_parms )
 *
 *  This callback function shows all static parameters.
 *****************************************************************************
/

XtCallbackProc cbr_show_parms ( widget, closure, calldata )

    Widget  widget;                 /* Set to the widget which initiated this callback
                                     * function.
                                     */

    caddr_t closure,                /* Callback specific data.  This parameter is not
                                     * used by this function.
                                     */

            calldata;               /* Specifies any callback-specific data the widget
                                     * needs to pass to the client.  This parameter is
                                     * is not used by this function.
                                     */
{
    D(printf("\nCBR_SHOW_PARMS\n"));
/*
 *  Display the already created popup by managing the main child within the shell.
 */

    XtManageChild ( parm_popup );

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( kill_job )
 *
 *  This function initiates termination of a selected job. It is called from
 *  cbr_kill_job to actually do the termination processing.
 *****************************************************************************/

int kill_job ( ls, index )

    struct ls_shm_str   *ls;

    int                 index;
{
    static char             f[] = "kill_job";

    struct ls_job_kill_str  job;

    D(printf("\nKILL_JOB\n"));
/*
 *  Initialize the job kill request structure. Add the process ID, the job source,
 *  and the job location.
 */

    job.jk_type = TYPE_KILL_JOB;
    job.jk_pid  = ls->ls_jobs[index].job_pid;
    strcpy ( job.jk_source,   ls->ls_jobs[index].job_source   );
    strcpy ( job.jk_location, ls->ls_jobs[index].job_location );

/*
 *  Send the request to the local scheduler. The scheduler will route the request if
 *  the job is on another host.
 */

    if ( msgsnd ( ls->ls_mq_id, (struct msgbuf *)&job, sizeof ( struct ls_job_kill_str ),
                  0 ) == -1 ) {
        display_message ( top, MSG_ERROR, "Could not write to message queue" );
        return ( err_log ( f, "Could not write to message queue", -1, errno ) );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
*
 * MODULE NAME AND FUNCTION ( ok_to_add_host )
 *
 *  This function determines converts a selected host item in the hosts list to an index
 *  into the host table in shared memory. It also performs error checking.
 *****************************************************************************
/

int ok_to_add_host ( host )

    char    *host;
{
    register int    i;

    D(printf("\nOK_TO_ADD_HOST\n"));
/*
 *  Verify that a selection was actually made. Display a warning if none was selected.
 */

    if ( *host == 0 ) {
        display_message ( top, MSG_WARN, "No host selection made" );
        return ( -1 );
    }

/*
 *  Find the index of the selected host in the host table in shared memory.
 */

    for ( i = 0; i < ls->ls_num_hosts; i++ )
        if ( strcmp ( host, ls->ls_hosts[i].host_name ) == 0 )
            break;

/*
 *  A match should always be found, but if not, display a message and return.
 */

    if ( i == ls->ls_num_hosts ) {
        display_message ( top, MSG_WARN, "Selected host not found in host table" );
        return ( -1 );
    }

/*
 *  Verify that the host is not already being displayed.
 */

    if ( disp_hosts[i].dh_shell_widget ) {
        display_message ( top, MSG_WARN, "Selected host is already displayed" );
        return ( -1 );
    }

/*
 *  Return index to the host in shared memory.
 */

    return ( i );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( schedule_job )
 *
 *  This function actually schedules a job.
 *****************************************************************************/

int schedule_job ( ls, job_buffer, total, job_name )

    struct ls_shm_str       *ls;

    struct ls_job_sched_str *job_buffer;

    int                     total;

    char                    *job_name;
{
    static char             f[] = "schedule_job";

    D(printf("\nSCHEDULE_JOB\n"));
/*
 *  Copy the job name into the job buffer structure.
 */

    strcpy ( job_buffer->js_name, job_name );

/*
 *  Send the job request to the local scheduler. The local scheduler will route
 *  a request to a remote host if necessary.
 */

    if ( msgsnd ( ls->ls_mq_id, (struct msgbuf *)job_buffer, total, 0 ) == -1 ) {
        display_message ( top, MSG_ERROR, "Could not write to message queue" );
        return ( err_log ( f, "Could not write to message queue", -1, errno ) );
    }

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( set_priority )
 *
 *  This function initiates setting of the priority on a job. It is called from
 *  cbr_set_priority to actually do the processing.
 *****************************************************************************/

int set_priority ( ls, index, priority )

    struct ls_shm_str   *ls;

    int                 index,
                        priority;
{
    static char             f[] = "set_priority";

    struct ls_job_pri_str   job;

    D(printf("\nSET_PRIORITY\n"));
/*
 *  Initialize the job priority change structure. Add the process ID, the priority, the
 *  the job source, and the job location.
 */

    job.jp_type      = TYPE_SET_PRIORITY;
    job.jp_pid       = ls->ls_jobs[index].job_pid;
    job.jp_priority  = priority;
    strcpy ( job.jp_source,   ls->ls_jobs[index].job_source   );
    strcpy ( job.jp_location, ls->ls_jobs[index].job_location );

/*
 *  Send the request to the local scheduler. The scheduler will route the request if
 *  the job is on another host.
 */

    if ( msgsnd ( ls->ls_mq_id, (struct msgbuf *)&job, sizeof ( struct ls_job_pri_str ),
                  0 ) == -1 ) {
        display_message ( top, MSG_ERROR, "Could not write to message queue" );
        return ( err_log ( f, "Could not write to message queue", -1, errno ) );
    }

/*
 *  Normal return.
 */

    return ( 0 );
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION ( tmr_update )
 *
 *  This timer function performs all periodic processing required to keep lists up to
 *  date.
 *****************************************************************************/

XtTimerCallbackProc tmr_update ( client_data, id )

    caddr_t         client_data;    /* Character data passed to this callback function.
                                     * It is currently unused by this function.
                                     */

    XtIntervalId    *id;            /* Identifies the timer which caused this function to
                                     * be activated.
                                     */
{
    register int            i,
                            length;

    static int              last_job_upd  = 0,
                            last_host_upd = 0,
                            num_jobs      = 0,
                            num_hosts     = 0;

    static char             f[] = "tmr_update",
                            host_name[MAX_HOST_NAME + 1] = "";

    struct ls_message_str   message;
    struct ls_job_str       *ptr;

    char                    string[200],
                            *ctime ( );

    D(printf("\nTMR_UPDATE\n"));
/*
 *  Save hostname to make subsequent processing easier.
 */

    if ( *host_name == 0 )
        strcpy ( host_name, ls->ls_hosts[0].host_name );

/*
 *  Update the list of active hosts. This occurs only if a new host has been added
 *  or an existing hosts is no longer active. Check the last update time and if
 *  different, save the current update time, remove all hosts from the current list,
 *  add all active hosts to the list, and save the number of hosts displayed.
 */

    if ( ls->ls_last_host_upd != last_host_upd ) {
        last_host_upd = ls->ls_last_host_upd;

        for ( i = 0; i < num_hosts; i++ )
            XmListDeletePos ( hosts, 0 );
        for ( i = 0; i < ls->ls_num_hosts; i++ )
            XmListAddItem ( hosts, XmStringLtoRCreate (
                                        ls->ls_hosts[i].host_name, DC ), 0 );
        num_hosts = ls->ls_num_hosts;
    }

/*
 *  Update the list of active jobs. Again this only occurs if a new jobs has been
 *  added or an existing job has changed state. Check the update time and if different,
```

```
 *  save the current update time and delete all jobs in the list.
 */

    if ( ls->ls_last_job_upd != last_job_upd ) {
        last_job_upd = ls->ls_last_job_upd;

        for ( i = 0; i < num_jobs; i++ )
            XmListDeletePos ( jobs, 0 );

/*
 *      Update the list with all active jobs. For each entry in the job table refer-
 *      encing an active job, add an item to the list.
 */

        num_jobs = 0;

        for ( i = 0; i < MAX_JOBS; i++ )
            if ( ls->ls_jobs[i].job_pid != -1 ) {
                ptr = &ls->ls_jobs[i];
                sprintf ( string, JOB_FORMAT, ptr->job_pid,    ptr->job_priority,
                                              ptr->job_source, ptr->job_location,
                                              "Dummy",
                                              ctime ( &ptr->job_start ),
                                              ptr->job_name );
                XmListAddItem ( jobs, XmStringLtoRCreate ( string, DC ), 0 );
                num_jobs++;
            }
    }

/*
 *  Update the scale widget which displays the current system load.
 */

    XmScaleSetValue ( lscale, ls->ls_hosts[0].host_current_load );

/*
 *  For each host information popup, update the scale widget with the load of the
 *  corresponding host.
 */

    for ( i = 0; i < ls->ls_num_hosts; i++ )
        if ( disp_hosts[i].dh_shell_widget )
            XmScaleSetValue ( disp_hosts[i].dh_scale_widget,
                              ls->ls_hosts[i].host_current_load );

/*
 *  Begin a loop which retrieves each status message pending on the message queue.
 */

    errno = 0;
    while ( errno != ENOMSG ) {
        if ( ( length = msgrcv ( ls->ls_mq_id, (struct msgbuf *)&message, MAX_MESSAGE,
                                 TYPE_INFO, IPC_NOWAIT ) ) == -1 && errno != ENOMSG )
            err_log ( f, "Could not read status from queue", -1, errno );

/*
 *      If a message was retrieved, get the length of the message and add it to the
 *      end of the message text.
 */

        if ( errno == 0 ) {
            i = XmTextGetLastPosition ( status );
            XmTextReplace ( status, i, i, message.msg_data );
            i = XmTextGetLastPosition ( status );
            XmTextReplace ( status, i, i, "\n"             );
        }
    }

/*
 *  Reset the timeout value.
 */

    XtAddTimeOut ( TIMEOUT, tmr_update, NULL );

/*
 *  Normal return.
 */

    return;
}
```

```
/*****************************************************************************
 * MODULE NAME AND FUNCTION: ( cleanup )
 *
 *  This function is runs a job.
 *****************************************************************************/

int cleanup ( )
{}
```

```
##############################################################################
##
# Makefile for load sharing prototype processes.
##############################################################################
##

#
# Define the target which this file is to create.
#

TARGET0 = liblsui.a
TARGET1 = libls.a
TARGET2 = ls_init
TARGET3 = ls_net
TARGET4 = ls_status
TARGET5 = ls_scheduler
TARGET6 = ls_monitor

#
# Initialize include paths.
#

MASTER      = /home/project/2881/ls
BINDIR      = $(MASTER)/bin
LSINCDIR    = $(MASTER)/include
HASINCDIR   = $(MASTER)/has/Includes
INCDIRS     = -I$(LSINCDIR) -I$(HASINCDIR)
LIBDIR      = $(BINDIR)

#
# Define the libraries to search.
#

LIBRARIES       = -lls
LIBRARIES_MON   = -lls -llsui -lXm -lXt -lX11

#
# Define the compiler and linker flags.
#

CFLAGS          = -g $(INCDIRS) $(FLAGS)
LDFLAGS         = -g -L$(BINDIR)

#
# Define all objects which make up this target.
#

OBJS0   =\
        ls_uilib.o

OBJS1   =\
        ls_lib.o

OBJS2   =\
        ls_init.o

OBJS3   =\
        ls_net.o

OBJS4   =\
        ls_status.o

OBJS5   =\
        ls_scheduler.o

OBJS6   =\
        ls_monitor.o

#
# Make the targets.
#

all:        $(TARGET0) $(TARGET1) $(TARGET2) $(TARGET3) $(TARGET4) $(TARGET5) $(TARGET6)

$(TARGET0): $(OBJS0)
    ar rv $(LIBDIR)/$@ $(OBJS0)
    ranlib $(LIBDIR)/$@

$(TARGET1): $(OBJS1)
    ar rv $(LIBDIR)/$@ $(OBJS1)
    ranlib $(LIBDIR)/$@

$(TARGET2): $(OBJS2)
    $(CC) -o $(BINDIR)/$@ $(OBJS2) $(LDFLAGS) $(LIBRARIES)

$(TARGET3): $(OBJS3)
    $(CC) -o $(BINDIR)/$@ $(OBJS3) $(LDFLAGS) $(LIBRARIES)

$(TARGET4): $(OBJS4)
    $(CC) -o $(BINDIR)/$@ $(OBJS4) $(LDFLAGS) $(LIBRARIES)

$(TARGET5): $(OBJS5)
    $(CC) -o $(BINDIR)/$@ $(OBJS5) $(LDFLAGS) $(LIBRARIES)

$(TARGET6): $(OBJS6)
    $(CC) -o $(BINDIR)/$@ $(OBJS6) $(LDFLAGS) $(LIBRARIES_MON)

#
# Dependencies
#
```

```
#
# Set up to attach each primitive widget to a corner.
#

*leftAttachment:                    ATTACH_POSITION
*rightAttachment:                   ATTACH_POSITION
*topAttachment:                     ATTACH_POSITION
*bottomAttachment:                  ATTACH_POSITION

#
# Global font and color defaults.
#

*fontList:                  -adobe-courier-medium-r-normal--14-140-75-75-m-90-iso885
9-1
*topShadowColor:                    white
*background:                        lightblue
*XmScale.showValue:                 TRUE

#
# Set up correct behavior on scrolled list widgets.
#

*jobs.scrollBarDisplayPolicy:       AS_NEEDED
*hosts.scrollBarDisplayPolicy:      AS_NEEDED
*jobs.listSizePolicy:               RESIZE_IF_POSSIBLE
*hosts.listSizePolicy:              RESIZE_IF_POSSIBLE

#
# Set the overall size of popups. This is to size the host popup, the
# name of which is dynamic ('hostname' Parameters).
#

*minWidth:                          300
*minHeight:                         200

#
# Set the size of the main window.
#

*ls_monitor.minWidth:               950
*ls_monitor.minHeight:              600

#
# Set values for the popup used to set the priority.
#

*Set Priority.main.width:           300
*Set Priority.main.height:          130

*Set Priority*label.leftPosition:   1
*Set Priority*label.rightPosition:  99
*Set Priority*label.topPosition:    1
*Set Priority*label.bottomPosition: 15

*Set Priority*scale.orientation:    HORIZONTAL
*Set Priority*scale.processingDirection:MAX_ON_RIGHT
*Set Priority*scale.minumum:        0
*Set Priority*scale.maximum:        20
*Set Priority*scale.leftPosition:   1
*Set Priority*scale.rightPosition:  99
*Set Priority*scale.topPosition:    16
*Set Priority*scale.bottomPosition: 55

*Set Priority*sep.leftPosition:     0
*Set Priority*sep.rightPosition:    100
*Set Priority*sep.topPosition:      60
*Set Priority*sep.bottomPosition:   63

*Set Priority*ok.leftPosition:      10
*Set Priority*ok.rightPosition:     30
*Set Priority*ok.topPosition:       70
*Set Priority*ok.bottomPosition:    90

*Set Priority*cancel.leftPosition:  40
*Set Priority*cancel.rightPosition: 60
*Set Priority*cancel.topPosition:   70
*Set Priority*cancel.bottomPosition:90

*Set Priority*help.leftPosition:    70
*Set Priority*help.rightPosition:   90
*Set Priority*help.topPosition:     70
*Set Priority*help.bottomPosition:  90

#
# Values for the main window frames.
#

*sframe.leftPosition:               1
*sframe.rightPosition:              50
*sframe.topPosition:                1
*sframe.bottomPosition:             25

*hframe.leftPosition:               51
*hframe.rightPosition:              99
*hframe.topPosition:                1
*hframe.bottomPosition:             25

*jframe.leftPosition:               1
*jframe.rightPosition:              99
*jframe.topPosition:                26
*jframe.bottomPosition:             70

*tframe.leftPosition:               1
*tframe.rightPosition:              99
*tframe.topPosition:                71
*tframe.bottomPosition:             99

*pframe.leftPosition:               1
*pframe.rightPosition:              99
*pframe.topPosition:                1
*pframe.bottomPosition:             99

#
# Values for the scale widget labels and scales. Note that scale 0 is never
# sensitive. Scales 1 and 2 are only sensitive in the main window.
#

*scale0.sensitive:                  FALSE
*hostform.scale1.sensitive:         FALSE
*hostform.scale2.sensitive:         FALSE

*sep0.orientation:                  VERTICAL
*sep1.orientation:                  VERTICAL

*slabel0.leftPosition:              1
*slabel0.rightPosition:             32
*slabel0.topPosition:               1
```

```
*slabel0.bottomPosition:                10

*scale0.leftPosition:                   1
*scale0.rightPosition:                  32
*scale0.topPosition:                    12
*scale0.bottomPosition:                 99

*sep0.leftPosition:                     32
*sep0.rightPosition:                    33
*sep0.topPosition:                      0
*sep0.bottomPosition:                   100

*slabel1.leftPosition:                  34
*slabel1.rightPosition:                 65
*slabel1.topPosition:                   1
*slabel1.bottomPosition:                10

*scale1.leftPosition:                   34
*scale1.rightPosition:                  65
*scale1.topPosition:                    12
*scale1.bottomPosition:                 99

*sep1.leftPosition:                     65
*sep1.rightPosition:                    66
*sep1.topPosition:                      0
*sep1.bottomPosition:                   100

*slabel2.leftPosition:                  67
*slabel2.rightPosition:                 99
*slabel2.topPosition:                   1
*slabel2.bottomPosition:                10

*scale2.leftPosition:                   67
*scale2.rightPosition:                  99
*scale2.topPosition:                    12
*scale2.bottomPosition:                 99

#
# Values for host list label and list.
#

*hlabel.leftPosition:                   1
*hlabel.rightPosition:                  99
*hlabel.topPosition:                    1
*hlabel.bottomPosition:                 10

*hostsSW.leftPosition:                  1
*hostsSW.rightPosition:                 99
*hostsSW.topPosition:                   11
*hostsSW.bottomPosition:                99

#
# Values for the job list label and list.
#

*jlabel.alignment:                      ALIGNMENT_BEGINNING
*jlabel.leftPosition:                   1
*jlabel.rightPosition:                  99
*jlabel.topPosition:                    1
*jlabel.bottomPosition:                 5

*jobsSW.leftPosition:                   1
*jobsSW.rightPosition:                  99
*jobsSW.topPosition:                    6
*jobsSW.bottomPosition:                 99

#
# Values for status label and text.
#

*lstatus.leftPosition:                  1
*lstatus.rightPosition:                 99
*lstatus.topPosition:                   1
*lstatus.bottomPosition:                8

*statusSW.leftPosition:                 1
*statusSW.rightPosition:                99
*statusSW.topPosition:                  9
*statusSW.bottomPosition:               99

#
# Values for parameter values in static parameters popup.
#

*Static Parameters.minWidth:            450
*Static Parameters.minHeight:           200

*statintl.leftPosition:                 1
*statintl.rightPosition:                39
*statintl.topPosition:                  1
*statintl.bottomPosition:               19

*statintt.leftPosition:                 40
*statintt.rightPosition:                48
*statintt.topPosition:                  1
*statintt.bottomPosition:               19

*advintl.leftPosition:                  1
*advintl.rightPosition:                 39
*advintl.topPosition:                   21
*advintl.bottomPosition:                39

*advintt.leftPosition:                  40
*advintt.rightPosition:                 48
*advintt.topPosition:                   21
*advintt.bottomPosition:                39

*heartintl.leftPosition:                1
*heartintl.rightPosition:               39
*heartintl.topPosition:                 41
*heartintl.bottomPosition:              59

*heartintt.leftPosition:                40
*heartintt.rightPosition:               48
*heartintt.topPosition:                 41
*heartintt.bottomPosition:              59

*chkhintl.leftPosition:                 1
*chkhintl.rightPosition:                39
*chkhintl.topPosition:                  61
*chkhintl.bottomPosition:               79

*chkhintt.leftPosition:                 40
*chkhintt.rightPosition:                48
*chkhintt.topPosition:                  61
*chkhintt.bottomPosition:               79

*hosttimel.leftPosition:                1
```

```
*hosttimel.rightPosition:               39
*hosttimel.topPosition:                 81
*hosttimel.bottomPosition:              99

*hosttimet.leftPosition:                40
*hosttimet.rightPosition:               48
*hosttimet.topPosition:                 81
*hosttimet.bottomPosition:              99

*lowvalminl.leftPosition:               51
*lowvalminl.rightPosition:              89
*lowvalminl.topPosition:                1
*lowvalminl.bottomPosition:             19

*lowvalmint.leftPosition:               90
*lowvalmint.rightPosition:              98
*lowvalmint.topPosition:                1
*lowvalmint.bottomPosition:             19

*lowvalmaxl.leftPosition:               51
*lowvalmaxl.rightPosition:              89
*lowvalmaxl.topPosition:                21
*lowvalmaxl.bottomPosition:             39

*lowvalmaxt.leftPosition:               90
*lowvalmaxt.rightPosition:              98
*lowvalmaxt.topPosition:                21
*lowvalmaxt.bottomPosition:             39

*highvalminl.leftPosition:              51
*highvalminl.rightPosition:             89
*highvalminl.topPosition:               41
*highvalminl.bottomPosition:            59

*highvalmint.leftPosition:              90
*highvalmint.rightPosition:             98
*highvalmint.topPosition:               41
*highvalmint.bottomPosition:            59

*highvalmaxl.leftPosition:              51
*highvalmaxl.rightPosition:             89
*highvalmaxl.topPosition:               61
*highvalmaxl.bottomPosition:            79

*highvalmaxt.leftPosition:              90
*highvalmaxt.rightPosition:             98
*highvalmaxt.topPosition:               61
*highvalmaxt.bottomPosition:            79

!
! Accelerators
!

*Exit.accelerator:                      <Key>F1
*Exit.acceleratorText:                  F1
*Run Job.accelerator:                   <Key>F2
*Run Job.acceleratorText:               F2
*Kill Job.accelerator:                  <Key>F3
*Kill Job.acceleratorText:              F3
*Set Priority.accelerator:              <Key>F4
*Set Priority.acceleratorText:          F4
*Show Host.accelerator:                 <Key>F5
*Show Host.acceleratorText:             F5
*Apply Limits.accelerator:              <Key>F6
*Apply Limits.acceleratorText:          F6
*Show Parameters.accelerator:           <Key>F7
*Show Parameters.acceleratorText:       F7
*Help.accelerator:                      <Key>F8
*Help.acceleratorText:                  F8
```