NUSC Technical Document 6998
1 April 1991

Operational Concept Document for the Next-Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

Operating Systems Standards Working Group (OSSWG)
Compiled by D. P. Juttelstad (NUSC)

Naval Underwater Systems Center
Newport, Rhode Island · New London, Connecticut

PREFACE

This report was funded under NUSC Project No. A45146, "Next-Generation Computer Resources (NGCR)." The sponsoring activity is the Space and Naval Warfare Systems Command, through the work of the Operating Systems Standards Working Group (OSSWG). The OSSWG management structure is as follows:

NGCR Program Manager, H. Mendenhall (SPAWAR-324)

NGCR OSSWG Cochairman, LCDR R. Voigt (SPAWAR-324)

NGCR OSSWG Cochairman, R. Bergman (NOSC)

Approach Subgroup Chairman, S. Howell (NSWC)

Exploration & Technology Subgroup Chairman, D. Juttelstad (NUSC)

Standards Evolution Subgroup Chairman, J. Oblinger (NUSC)

This document was developed over a 2-year period by a team from the Navy, other sections of the Government, private industry, and academia, who are experts in the field of computer operating systems. Only a few of the Navy participants were actually funded to directly participate in this process. The superb accomplishments of the joint working group and its ability to complete this effort in a relatively brief time span were a direct result of the total dedication of all participants to the project. The outstanding contributions of all the volunteers in this process are particularly noted and appreciated.

Special thanks are expressed to U.S. industry and academia for their staunch support and participation in this working group. Their continued support and involvement are strongly solicited.

REVIEWED AND APPROVED: 1 APRIL 1991

P. A. La Breeque
Head, Combat Control Systems Department

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
1 April 1991

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
Operational Concept Document for the Next-Generation Computer Resources (NGCR) Operating Systems Interface Standard Baseline

5. FUNDING NUMBERS

6. AUTHOR(S)
Operating Systems Standards Working Group (OSSWG)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Underwater Systems Center
Newport Laboratory
Newport, RI 02841

8. PERFORMING ORGANIZATION REPORT NUMBER
TD 6998

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Space and Naval Warfare Systems Command
(SPAWAR-324)
Washington, DC 20363

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Next-Generation Computer Resources (NGCR) Operating Systems Standards Working Group (OSSWG) conducted a survey of existing operating systems and operating systems interface standards to establish a baseline for the NGCR operating system interface. This document will identify the requirements for services, interfaces, and protocols to be supported by the operating systems interface.

14. SUBJECT TERMS
Next Generation Computer Resources
Operating Systems Interface
Operational Concept Document

15. NUMBER OF PAGES
155

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

OPERATIONAL CONCEPT DOCUMENT FOR THE NEXT-GENERATION COMPUTER RESOURCES (NGCR) OPERATING SYSTEMS INTERFACE STANDARD BASELINE

1. SCOPE

1.1 IDENTIFICATION

This is the Operational Concept Document (OCD) for the Next-Generation Computer Resources (NGCR) Operating Systems Interface (OSIF) Standard. The format of this document is based on MIL-STD 2167A, "Defense System Software Development," Data Item Description DI-MCCR-80023.

This OCD incorporates material taken from references 1 through 9 (see section 2).

1.2 PURPOSE

The purpose of this OCD is to present the operational concept of the OSIF, which was derived through a consensus among Navy agencies, industry, and academia. This OCD will identify the requirements for services, interfaces, and protocols to be supported by the OSIF.

The purpose of the OSIF is to establish a set of interfaces that controls the operations of all computing system hardware and software elements of a platform. These elements are coordinated in a uniform manner that is consistent with the mission of the platform. The OSIF is an interface specification; it does not specify an operating system implementation. The objective of the baseline specification is to select a commercially available OSIF or family of interface baseline requirements and modify them as little as possible.

1.3 INTRODUCTION

The NGCR program is designed to fulfill the Navy's need for standard computing resources while allowing it to take advantage of commercial products and investments, and to field new technology advances more quickly and effectively. The program revolves around the definition and selection of standards, one of which is the establishment of the OSIF. The effort to establish such an interface standard, initiated at the start of 1989, draws on industry, academic, and military expertise. An initial OSIF standard is expected in 1993; the final standard is expected to be usable in the procurement of Navy systems in 1996.

It is required that the NGCR OSIF standard be Ada-oriented, real-time, distributed/networked, scalable, multilevel secure, reliable, and realizable on heterogeneous processors.

2. APPLICABLE DOCUMENTS

1. "NGCR Development Options Paper," SPAWAR ltr to NCGR OSSWG, Ser. 324.253, 30 October 1987.

2. "NGCR Program Master Plan," SPAWAR-324 Informal Document, 12 June 1989.

3. "NGCR Operating Systems Standards Working Group (OSSWG) Technology Report," SPAWAR-324 Informal Document, Version 0.1, 1 August 1989.

4. "NGCR Operating Systems Standards Working Group (OSSWG) Reference Model," SPAWAR-324 Informal Document, Version 1.02, 6 August 1989.

5. "Department of Defense Trusted Computer Systems Evaluation Criteria," DoD 5200.28-STD, Department of Defense, Washington, DC, December 1985.

6. "The Challenge of Ada Runtime Environments White Paper," Ada Runtime Environment Working Group, MRTSI Taskforce, Association for Computing Machinery Special Interest Group for Ada, October 1988.

7. "OSSWG Architecture Model for Embedded Operating Systems," SPAWAR-324 Informal Document, 15 June 1989.

8. IEEE POSIX Standard 1003 and future Standards, Institute of Electrical and Electronic Engineers.

9. "Tactical Digital Standard (TADSTAND) A, Basic Policies, Procedures, and Definitions for Mission-Critical Computer Resources (MCCR)," SPAWAR-324 Informal Document, Draft Revision 1, November 1989.

3. MISSION

3.1 MISSION NEED REQUIREMENTS

Historically, the Navy acquisition and budgeting processes have extended over such long time periods that, on deployment, the technology in new Navy standard computers is often obsolete relative to that of current commercial technology. Even when preplanned product improvements are programmed, budget uncertainties and a lengthy acquisition process delay introduction of these improvements.

This situation has led to an ever-increasing number of waivers to allow the use of nonstandard computers in Navy weapon systems. It has also led to the introduction of commercial products directly into military computers, which has generated a drive to introduce commercial, nonstandard computer products into mission-critical applications. Typical mission-critical applications include aircraft, surface, weapons, subsurface, shore-based facilities, etc.

Because of these constraints in the current acquisition and implementation procedure, the NGCR program has chosen a joint industry/Navy standards definition and development approach. This method is based on an open-system architecture (OSA) approach that is commercially based and supplemented where necessary to achieve specific Navy requirements. The incentives for rapid acceptance of the standard include wide access to the underlying commercial technology; wide access to an open Navy market; reduced cost and risk of research, development, test, and evaluation (RDT&E); and increased competition among vendors.

The NGCR OSIF Standard is not a design of the operating system, but is a specification of interfaces to the operating system.

The following subsections highlight the requirements of the NGCR OSIF Standard program.

3.1.1 *Portability*

There is a need for application software portability that provides sufficient functionality to accommodate a broad range of application requirements. Architecture-independent services will allow an NGCR system to interface with non-NGCR resources.

3.1.2 *Reusability*

The OSIF shall support reusability. Reusability for new developments will take place (at least) at the source code level, enabling developers to reuse portions of applications source code (and/or other pertinent aspects such as design and tests) in the generation of other applications.

3.1.3 *Training*

The OSIF shall allow for a standard operations and maintenance curriculum to be developed. This framework will be directly applicable to all OSIF baseline standard users, thus reducing system-specific training costs.

3.1.4 *Nondevelopmental Items and Software*

A nondevelopmental item (NDI) is defined as a deliverable item that meets acquisition requirements but is not developed under contract, and can be provided by the contractor, by the Government, or by a third party. These items include commercial products, products developed and used by another service or Government agency, and products developed and used by other countries.

Nondevelopmental software (NDS) is defined as deliverable computer software not developed under contract but provided by the contractor, by the Government, or by a third party. NDS may be referred to as reusable software, as Government-furnished software, or as commercially available software.

The OSIF shall provide the ability to maximize the use of NDIs and NDSs and minimize proprietary development items. An incremental process of introducing NDIs and NDSs conforming to the NGCR OSIF Standard is projected.

3.1.5 *Multilevel Security*

The NGCR program shall provide standards that support the security needs of the Navy. The NGCR OSIF Standard should not preclude meeting the operational requirements for any security mode of operation (i.e., dedicated, system high, compartmented, multilevel) for either a local or distributed operating system. Meeting the security needs of a project is a particularly difficult issue to resolve when coupled with the other requirements of tactical systems. Operating systems developed to the NGCR OSIF Standard shall maximize the protection of system integrity from inadvertent or malicious misuse according to a system's needs. Some systems may require multiple concurrent levels of security within a node, as well as across the distributed system. The security mechanism should also conform to available and evolving Department of Defense (DoD) security standards as appropriate.

The implementation and support services of the operating system in accordance with the NGCR OSIF Stardard must meet the security requirements as specified in the "Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, of December 1985 (reference 5). A risk analysis and security concept of operations should be defined for each mission to determine the appropriate class of requirements for the mission.

3.1.6 *Reliability, Maintainability, Adaptability, and Availability*

The NGCR OSIF Standard shall provide for the development of operating system services that support fault-tolerant system requirements. This

includes fault tolerance and prediction interface capabilities. System developers shall be able to specify system monitoring capabilities and system reconfiguration of resources in response to specified error data.

3.1.7 *Real-Time Performance*

The acceptable performance of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced. The Navy real-time system design often distinguishes between hard and soft real-time systems.

Hard real-time systems are those systems in which it is absolutely imperative that responses occur within the specified deadline. Soft real-time systems are those where response times are important, but the system will perform acceptably if deadlines are missed within statistical bounds.

In a hard or soft real-time system, the computer is usually interfaced directly to some physical equipment and is dedicated to monitoring or controlling the operation of that equipment. A key feature of all these applications is the role of the computer as an information-processing component within a larger system. These types of Navy systems are referred to as "embedded computer systems."

It is crucial that the NGCR OSIF Standard provide the necessary interfaces to support the development and testing of embedded computer systems. The interfaces must provide system developers access to operating system services that are needed to meet application-defined time constraints. If there is a possibility that the user-defined timing constraints may not be met, the OSIF shall provide for notification of this fact to the application software in the form of a failure prediction as well as a failure occurrence.

3.1.8 *Language Support*

Although the NGCR OSIF Standard compliant operating system may be implemented in various languages, it shall support, at a minimum, the execution of programs written in Ada and C.

3.1.9 *Distributed System Support*

The OSIF shall support resource allocation/scheduling for systems consisting of multiple computing resources including other NGCR components. The OSIF shall also provide services to support the scheduling of two or more processes on a single processor within a multiprocessor system.

3.1.10 *Heterogeniety*

The OSIF shall provide for operating system implementation on heterogeneous processors and the integration of heterogeneous processors in a single-system environment.

3.2 PRIMARY MISSION

The primary mission of the NGCR OSIF Standard is to provide a standard operating system interface that both meets the needs of Navy applications and enjoys commercial acceptance. This will allow the Navy to promote open competition and to use commercially available system components.

3.3 SECONDARY MISSION

The secondary mission of the OSIF is to support all Navy computer systems. As an additional secondary mission objective, the OSIF will promote the following benefits:

- Reduced operating system development costs.
- Reduced operation and maintenance costs for items such as training and documentation.
- Avoidance of replication of Navy RDT&E cost for separate projects to develop similar capabilities.
- More effective system integration.
- Increased software reusability.

3.4 OPERATIONAL ENVIRONMENT

The operational environment support includes services that assist in the dissemination of the NGCR OSIF Standard, evolution of the standard, and consultation in the use of the standard. Operational environment support shall be provided to program managers, acquisition managers, system designers and developers, integrators and users, and others in the selection of an OSIF implementation that meets the needs of a specific program. The benefits of this operational environment support include reduced program cost, quality assurance, and independent validation and verification.

The NGCR Program Management Office shall identify the approach to validating and using the OSIF. This approach shall ensure a standard across all Navy systems to promote ease of acquisition, maintenance, systems design, and software development. The NGCR Program Management Office shall identify agents responsible for establishing and executing relevant policy. Through validation procedures, a list of NGCR-accredited products will be maintained. The NGCR Program Management Office shall establish a plan for maintaining a list of accredited products. Included in this plan shall be other practical aspects related to selection and/or implementation of the standard(s).

The NGCR Program Management Office shall publish a companion handbook for scalability of the NGCR OSIF Standard. The degree to which the standard may be scaled shall be specified in this companion handbook.

3.5 SUPPORT ENVIRONMENT

The Navy will establish and maintain a military standard that describes the OSIF. The military standard might be a document that references and invokes one or several national, commercial, or international standard(s) (e.g., the Institute of Electrical and Electronic Engineers (IEEE), the American National Standards Institute (ANSI), the International Standards Organization (ISO)). These will comprise the NGCR OSIF Standard. This allows published standards to meet a range of Navy application needs while supporting them with widely used commercial technology.

The NGCR Program Management Office will be responsible for establishing and carrying out procedures during definition and after implementation of the standard to maintain and upgrade the OSIF as a "living document." After publication, the NGCR Program Management Office will regularly monitor the standard by using the working group process. They will participate in all change meetings that might be held to ensure the Navy's computer technology needs are continually and adequately addressed. Application of commonly accepted standards to Navy systems will leverage commercial R&D investments and take advantage of a commercial R&D investment and a competitive commercial market. Once published, these standards will provide significant market incentives for multiple private-product developments. A list of these products may be used for acquisition purposes.

When it becomes necessary to redefine the standard because technology advances beyond the scope of the OSIF, the NGCR Program Management Office will reform the Operating Systems Standards Working Group (OSSWG) from industry and Navy representatives. Existing procedures will be maintained to reach a consensus from the participants on a new or updated NGCR OSIF Standard.

4. INTERFACE STANDARD FUNCTIONS AND CHARACTERISTICS

4.1 OPERATING SYSTEM SERVICES

This section describes the general requirements and the major groups of operating system services that are required of the NGCR OSIF. Not all of these services require an explicit programming interface, nor will all service definition sets for an operating system or implementations of those definition sets group services in the same way. Many of the application domains described in section 4.2 will need only a subset of the services listed here. Because there are dependencies and interactions among the service groups, a means to identify compliant subsets for specific application domains is required. The implementations of the subsets appropriate for different application domains will also usually have to make different kinds of tradeoffs with respect to differing aspects of performance. Thus, even in some cases where the subsets of services required for different domains are very similar, different implementations may be required to support the performance needed.

This section provides an overview of the general requirements and service groups. More details are found in the appendixes.

4.1.1 *General Requirements*

This section outlines the general goals of the NGCR OSIF Standard and the characteristics that the interface standards must possess. They apply to all services defined by the set of NGCR OSIF Standards that fulfill the system needs expressed in this OCD.

The goals are to provide the following capabilities for application programs:

1. Interoperability - the ability of two applications to share data.

2. Portability - the ability to move an application from one implementation of the OSIF to another with minimal changes to the source code.

3. Reusability - the ability to reuse portions of one application's source code or other pertinent aspects (e.g., design, tests) in the generation of another application.

4. Compatibility - the general ability of two applications to coordinate with one another in their operation, even if they were not originally designed to do so.

5. Maintainability - the qualities that improve the ability to maintain the application.

6. Time criticality - the ability to support systems in which time is a critical and controlled parameter.

The following are the general characteristics that the NGCR OSIF Standard must possess:

1. Uniformity - The standards must be based on one or a few well-defined consistent conceptual models. They must be expressed in a uniform, precise, and unambiguous fashion.

2. Independence - The standards must be expressed in such a fashion that they are independent of specific languages and hardware entities.

3. Scope - The set of standards must cover the total application range of interest.

4. Customization - The standards must support the selection of subsets or the addition of new capabilities to meet specific system requirements.

4.1.2 *Architecture-Dependent Services*

These services allow an NGCR resource to interface with non-NGCR resources such as computers, networks, and operating systems. This will facilitate portability and technology insertion and provide the ability for a system using NGCR-compliant components to interface with non-NGCR-compliant systems and components.

4.1.3 *Capability and Security Services*

The services specified in the NGCR OSIF Standard represent the OSIF functionality and service capability from a security perspective. Security is only represented from an interface perspective, and not from the perspective of the operating system. In addition, it is generally understood that information security should be considered from a system perspective. This involves a trusted computing base, which consists of the interface specification, the underlying security kernel, trusted application processes, and necessary system-level assurances. The level of security required for a particular mission will be based on the risk assessment and the mission's security mode of operation. System security requires a definition of the mode of operation, a statement of the policy required to achieve the desired level of security, and a reference monitor to enforce the security policy. The reference monitor includes services to be offered by the operating system to control the usage of the resources and the protection of classified data. The system, which includes the reference monitor and other trusted software, will be required to process and protect classified data from inadvertent or malicious misuse in accordance with the defined policy.

The services must support the following:

- Prevention of unauthorized access
- Prevention of data compromise

- Prevention of service denial
- Security administration
- Data integrity.

Capability services attach operation lists that limit the ability of the processes to act on resource objects. This is to ensure that the resources are not misused. Access to resources can be protected by services using capability lists as well as access lists, lock/key mechanisms, global tables, or through dynamic protection structure services.

4.1.4 *Data Interchange Services*

This set of services provides data conversion among different data representations used by separate components of the system.

4.1.5 *Event and Error Management Services*

These services provide a common facility for the generation, communication, logging, and control of asynchronous events for the system hardware components and application processes. Major uses of event services include reporting error conditions and providing an indication of some asynchronous event to the application processes by the device drivers or the operating system.

4.1.6 *File Services*

These services allow the applications to create and manipulate permanent storage for data. Such permanent storage is typically on disk or tape. The data are stored as files, and the files are organized in directories. Files are managed and accessed through logical names by the many system components that use the files, such as the application, target system operator, and project support environment (PSE). The major types of services are as follows:

1. Naming and directory services - These services allow the access of files and directories through logical names rather than through physical addressing conventions. These services will allow and control the sharing of files in a variety of ways. The directory services present a view or views of the directory structure to the application or target system operator.

2. File modification services - Services for files and directories include such things as the ability to read a file, to create a new file, and so on. These services may be very complex. For example, the access to read or to write may be direct (by record number), sequential (one record at a time), or indexed (by a tag). Real-time systems need special file services and implementations to ensure fast, predictable, and consistent performance in time-critical situations. The need for a known response time for a given input/-output (I/O) function drives the design and implementation of these files and services.

3. File support services - Additional services are required to support the physical devices on which the files and directories reside. These services include the dismounting/mounting, formatting, and partitioning of media.

4.1.7 *Generalized Input/Output Services*

These services provide a set of abstract and standard OSIFs for doing I/O to devices outside of the file I/O paradigm. All types of devices are to be covered (e.g., printers, disk drives, and analog-to-digital converters). The services, which may be used either synchronously or asynchronously (non-blocking), are as follows:

- Directory and naming services (discussed previously)
- Data transmission services
- Device control services.

4.1.8 *Networks and Communications*

These services involve the information exchange between the local processor nodes of an NGCR system. The following services are largely the application of general operating system service requirements to the network components:

1. Network control and status - These services provide authorized users with the capabilities to determine and control the status of network components and to control network working parameters.

2. Directory and naming services - These services allow the usage of system resources through logical names rather than through the actual hardware device naming conventions. Furthermore, they allow the resources of other processor nodes to be accessed by means of a logical name so that no knowledge of the resource's location is needed (the resource's location may change over time). The logical name to physical name relationship can be one to many, many to one, or many to many.

3. Interprocess communication - This service allows a local processor node's local operating system to request a procedure, a function, or a transaction to be performed on another processor node or logical resource and to communicate information among processes on different local nodes.

4.1.9 *Process Management Services*

Typically, the process management services are required to do the following:

1. Support multiple programs (i.e., where a program is an integrated set of processes) within a system.

2. Create a process and make it ready for execution.

3. Destroy a process and recover its resources.

4. Control the execution of processes.

5. Control the connection among processes, where a connection is a logical communication path among a set (two or more) of processes.

6. Provide the capability to query the state and capabilities of a process and to dynamically alter these.

4.1.10 *Project Support Environment Services*

The OSIF shall support the PSE in the development phase. During this process, there is a need for the PSE to communicate with the system under development. The operating system in the target will need to support that communication. These services may not be available at the application program interface (API), but may be accessed by means of a different interface. These services may also be removed from the system when it is deployed. The types of services include down-loading of compiled programs and data into the target system, uploading of program results and trace information to the PSE, and the interactive debugging by a developer on the PSE of an application running on the target system.

The OSIF shall provide PSE support for maintaining processed execution history information. Interfaces must allow for a user to define the execution history data requirements that the operating system must collect and return during software development and test.

4.1.11 *Reliability, Adaptability, and Maintainability Services*

The services supporting reliability, adaptability, and maintainability are often implied services in that there is not a direct interface to these services. Reliability and adaptability services deal with the need for the system to perform functions that the application requests in a timely manner. Reliability is the ability to correctly perform a job to completion; adaptability is the ability to change the system's logical makeup (or jobs to do) over time; and maintainability is the ability to keep the system in operating condition. A highly adaptable system can facilitate the reliability of an application's functions.

The following is a more detailed outline of these types of services:

1. Fault tolerance services - allow the system to react to the loss or incorrect operation of system components at various levels (hardware, logical, services, etc.). These services include

a. Fault tolerance services and event and error management services (these are closely related).

b. Fault detection services - concerned with determining when a fault has occurred in the system.

c. Fault isolation services - attempt to determine the component at fault and segregate the faulty component from the rest of the system.

d. Fault recovery services - attempt to bring the system into a consistent state.

e. Fault diagnosis services - analyze the attributes of a system fault and determine its cause.

f. Fault prediction services - involve the avoidance of faults before a failure in the system component occurs. Fault prediction frequently involves the logging of stressful conditions so that faults can be predicted and avoided.

g. Reconfiguration services - allow the system to substitute different resources to perform system functions such as substituting a new physical I/O channel to support a logical channel. Their use may be restricted to specially authorized processes concerned with system management.

2. Maintainability services - provide support for the maintenance of the embedded system. A major component of that support is the collection and logging of information about the operation of the system. Typical information to be logged includes

- Software and hardware errors during operation
- Processes that failed or almost failed to meet scheduled deadlines
- Performance metrics for system tuning
- Errors reported during startup self-testing.

4.1.12 *Resource Management Services*

These services are involved in the management of the system's resources, particularly memory. Resources include CPU, memory, I/O, and other physical devices.

Memory management services support the usage of the system's random access memory(s). These services supply a view of the memory or memories on the computer as seen by applications. They perform the proper mapping of virtual to physical memory by performing any swapping of memory pages needed in the process. Memory management services provide storage for processes, data migration, and initialization of the system.

The memory manager receives requests for services from the processes that allocate and de-allocate memory for process usage. The major services of

memory management fall into five categories: allocating physical memory, mapping of logical address to physical storage, sharing memory, extending memory (virtual storage), and protecting user information.

Resource management interfaces include services that control the access to system resources. System resources are any peripherals, communications links, common processes, etc., that may be asynchronously accessed by processes. Resource management also allows for queries about the status of these system resources.

4.1.13 *Synchronization and Scheduling Services*

Synchronization services coordinate in time the operations of other services, functions, processes, and/or resources. Services such as distributed voting and remote resource allocation will need to use these services to accomplish their required functionality. Synchronization services are needed for both the operation of the local processor operating system (LPOS) and the control of the distributed system. Synchronization services may need to use system monitoring services to adjust to system changes.

Scheduling services schedule or arbitrate the usage of various resources in the specific NGCR system, particularly the CPU. The scheduling services must be able to queue up requests to use a specific resource. This situation is made more complicated by the common need to schedule processes to run cyclically at a fixed period.

4.1.14 *System Initialization and Reinitialization Services*

System initialization includes starting the software completely, starting the attached hardware subsystem devices, doing subsystem and system self-tests, and initializing the data base completely.

System reinitialization includes restarting the software while using the existing data base information. The software may have to be reloaded and the data base may have been re-established by a system recovery. Attached hardware subsystems may also need to be reinitialized. Reinitialization should include a function to restart applications redistributed to other processors after a processor module failure. Within a processor, there should be a function to initialize applications in a system with the existing software but with the data base reinitialized. Also within a processor, there should be a function to restart the applications in a system with the existing software and data base retained.

4.1.15 *Time Services*

A number of time management services are to be supported by the operating system. The precision required by a given service will depend on system needs but will frequently be in the range of milliseconds to nanoseconds. These services are as follows:

1. Local time that provides date and time of day to a system-specified precision in a single or multiprocessor system. A variety of formats may be required.

2. Synchronization of local time among the components of a distributed system.

3. Synchronization of the local time of a system to an external time reference (e.g., Greenwich Mean Time).

4. Measurement of elapsed time.

5. Requests that a process be delayed for a specified elapsed time.

6. Requests that a process be delayed until a specific time.

7. Requests for process notification at a specific time or after a specified delay.

4.1.16 *Ada Language Support Services*

Although an operating system compliant with the NGCR OSIF Standard may be implemented in various languages, it must support the execution of programs written in Ada. This is required because the Ada language incorporates features that are accomplished by direct user calls to the operating system in other languages.

At a minimum, the operating system must provide primitive services that the Ada compiler's runtime library can use to implement the full semantics of the Ada language. For greater efficiency, some parts of the Ada runtime environment (ARTE) may be an integral part of the operating system implementation, even though the interface definition should not depend on that integration (reference 6).

4.2 APPLICATION DOMAINS FOR OSIF STANDARD FUNCTIONS

4.2.1 *Interactive Processing Domain – Ruby*

This application domain frequently will feature online transaction processing; off-the-shelf software products; networking to PCs, workstations, and other host environments; and background processing. The domain is characterized by strong requirements for data management, data reformatting, file services, generalized I/O, and resource management. By contrast, specific requirements for operating system support for languages and an interface to project support environments are low. This domain includes shore-based logistics systems, for example. In extant technology, implementations of these systems typically involve wide area networks of multiple, heterogeneous processors linked through gateways, etc.

4.2.2 *Special-Purpose Processing Domain – Opal*

This application domain consists of special-purpose dedicated processors, high data rates, and computationally intensive cyclic processing. The domain is characterized by strong requirements for event and error management, generalized I/O, and times services. By contrast, specific requirements for operating system support for data management, file system, and man-machine interface (MMI) are minimal. In extant technology, implementations of these systems typically involve one or more specialized processors such as might be found in signal-processing applications.

4.2.3 *Reliable Message Processing Domain – Amethyst*

This application domain consists of massive switching, store and forward, message-processing encryption, and error detection and recovery. The domain is characterized by strong requirements for security, fault tolerance, nuclear survivability, event and error management, networking and communications, scheduling and synchronization, and time management. By contrast, specific requirements for operating system support for PSEs are low. In extant technology, implementations of these systems involve processors on a given platform that must interface with networks that are widely distributed or with intraplatform networks. For example, this would include processors interfacing with global command, control, and/or intelligence systems.

4.2.4 *Embedded Processing Domain – Garnet*

This application domain consists of autonomous embedded processors with a wide spectrum of data rates and duty cycles. Overall, the domain does not put high demands on the operating system, and is characterized by strong requirements for language support, reliability, and availability. By contrast, specific requirements for operating system support for security, data management, file services, MMIs, and network and file communications are low. This application domain is exemplified by single processors embedded in missile warheads, torpedoes, and shipboard guns, for example.

4.2.5 *High Computation Domain – Topaz*

This application domain is characterized by high computational needs, interface to multiple sensors or controls or both, and support of interactive displays. Overall, the domain puts high demands on the operating system. It has strong requirements for operating system support for languages, data management, data reformatting, MMIs, and reliability and availability. These applications include major subsystems of shipboard systems such as navigation, ship control, or command systems. In extant technology, such applications typically include heterogeneous processors directly communicating on a near-continuous basis. One processor is typically a high-speed graphics processor.

4.2.6 *Mission-Critical Systems Domain – Emerald*

This application domain consists of mission-critical systems that are characterized by nuclear safety, command significance, and large ramifications of system failure. The domain has strong requirements for operating system support for security, reliability, and availability. By contrast, specific requirements for operating system support for files services are low. In extant technology, these applications are frequently embodied in single processors exercising centralized control over other processors or devices or both. The application includes (for example) processors that control the enabling, targeting, and firing of strategic weapons. These systems typically include requirements for access control and MMI management.

4.2.7 *Networked Processor Domain – Diamond*

This application domain consists of networked dedicated processors connected to multiple sensors, controls, and displays. The domain is characterized by strong requirements for operating system support for languages, hardware architecture dependencies, event and error management, reliability and availability, scheduling and synchronization, and time services. By contrast, specific requirements for operating system support for data management and file services are low. The domain typically includes multiple heterogeneous processors with particularized, dedicated functions. These processors are linked for cooperative interaction as might be found, for example, in avionics applications.

4.2.8 *Integrated Subsystems Domain – Sapphire*

This application domain consists of many cooperating subsystems that carry out mission-critical functionality. Overall, the domain puts high demands on the operating system, and is characterized by strong requirements for operating system support for languages, security, networking and communication, process management, PSE, reliability and maintainability, and time services. By contrast, specific requirements for operating system support for file services are low. This domain typically includes multiple platform-local networks of large numbers of heterogeneous processors. Examples include large tactical combat systems such as might be found aboard major surface ships and submarines.

5. RESPONSIBLE GOVERNMENT AGENCIES

The responsible Government agencies are as follows:

- Chief of Naval Operations (OPNAV)
- Space and Naval Warfare Systems Command (SPAWAR).

6. NOTES

6.1 GLOSSARY OF TERMS

The definitions of many of the terms provided in this section were taken from documents listed in section 2.

ADA RUNTIME ENVIRONMENT INTERFACE (ARTEI)

This is the interface between the application Ada programs and the runtime environment required by the Ada programming language. The ARTE may be implemented in three parts: (1) a part loaded with each application program from the Ada runtime library provided by the compiler vendor, (2) a part that is generated by the Ada compiler during the translation of the Ada program, and (3) a part that is a subset of the OSIFs. It is an issue as to how much of the necessary ARTE is part of the operating system, and how much should be provided by the compilation system of a compiler vendor. Most likely, some parts of this interface will be provided to allow applications to modify or "tune" the runtime system and other parts of the interface will be used only by compiler-generated code. Standardization of the interface to the Ada operating system (OS) services will allow the efficient integration of the ARTE and the LPOS. This interface is one aspect of the more general high-order language bin.

APPLICATION-LEVEL INTERFACE

This can be defined as

1. A shared boundary between an application program and system software.

2. A set of facilities that are provided for use by application programs.

3. A point at which independent systems or diverse groups interact; the device or system by which interaction at an interface is effected.

APPLICATION-LEVEL SERVICE

This can be defined as

1. That part of an operating system implementation that achieves the service requested when an interface is called.

2. That part of an operating system that is achieved by using a facility.

APPLICATION-LEVEL PROTOCOL

This can be defined as

1. A set of conventions or rules that govern the interactions of processes or applications within a computer system or network.

2. A set of rules that govern the operation of functional units to achieve communication.

APPLICATION SOFTWARE

This is software specifically produced for the functional use of a computer system, e.g., software for navigation, gun fire control, payroll, general ledger.

BINARY APPLICATION PROGRAM INTERFACE (BAPI)

This interface is the machine code-level calling sequences to the LPOS and the binary-level version of the source application program interface (SAPI). This is logically the same interface as the SAPI described later, but may require a separate standard if the compiler vendors are to be decoupled from the LPOS vendors. As an example of a BAPI, some machines use a set of software interrupt instructions to call operating system routines; the specific interrupt numbers and the specific conventions for parameter passing are part of the BAPI. Without a common interface at the binary level, a separate version of each compiler (or at least its code generator) may be required for each different implementation of the LPOS software, and code compiled by different compilers may not operate together. Note that even with standardization of the BAPI, there is (at best) a set of standards, one for each processor or processor family.

CENTRALIZED COMPUTER SYSTEM

A completely centralized design would be one in which all system functions were carried out in a single computer by a single process. Communication within the parts of the process is by the use of variables associated with it.

COMPUTER

This can be defined as

1. A functional unit that can perform substantial computation, including numerous arithmetic operations or logic operations, without intervention by a human operator during the run.

2. A functional programmable unit that consists of one or more associated processing units and peripheral equipment, is controlled by internally stored programs, and can perform substantial computation, including numerous arithmetic operations or logic operations, without human intervention.

CONFORMANCE

This is an adherence to a standard by an implementation.

DATA BASE KERNEL INTERFACE (DBKI)

Those systems that will require a multilevel, secure OS will require that some low-level parts of system-level components (LPOS, data base management system (DBMS), etc.) be fully integrated (i.e., be part of the trusted computer base (TCB)); therefore, the DBMS must have efficient access to the TCB. If the TCB is integrated with the OS kernel, then that kernel must provide the needed interfaces for the DBMS. The DBKI is the interface that the DBMS uses to access the TCB or OS kernel. Without a DBKI built into the LPOS, a DBMS may have to be built "on top of" the OS, which could lead to poor DBMS performance. If the DBKI is of general usefulness to application programs, then this interface may simply be a subset of the application program interface (SAPI and BAPI) to the operating system and, therefore, not a separate interface. It could also be a separate, special-purpose interface, which is provided only for those systems that use a DBMS.

DISTRIBUTED

The terms "centralized," "federated," and "distributed" represent ill-defined points along a continuum of computer system design possibilities. The terms are an attempt to express the manner in which and degree to which the components of the system are connected and work together. They, nonetheless, represent useful concepts in describing the overall architecture of a computer system. A real system of any complexity will frequently be usefully considered as being made up of subsystems that exhibit a certain degree of this property, while the real system itself has a different degree of the property. A completely centralized design would be one in which all system functions were carried out in a single computer by a single process. Communication within the parts of the process is by the use of variables associated with it.

A distributed design has been further defined as follows:

1. Multiplicity of general-purpose resource components, both physical and logical, that can be assigned to specific tasks on a dynamic basis.

2. Physical distribution of resources (components) with their interaction through a communication network.

3. Cooperative autonomy characterizing the operation and interaction of both logical and physical resources.

DISTRIBUTED OPERATING SYSTEM

This is a single homogeneous operating system implemented for an entire network.

FAMILY MEMBERS

These are defined as a set of OSIF standards, with each member appropriate to a specific application domain.

FAULT

This is a condition that prevents the continued predictable use of a hardware resource.

FEDERATED COMPUTER SYSTEM

A federated computer system design is one in which the total system function is divided into subfunctions, each of which is identified with a subsystem and uses one or more computers to carry out its work.

FUNCTIONAL UNIT

This is an entity of hardware, software, or both, capable of accomplishing a specified purpose.

GRAPHICS KERNEL INTERFACE (GRKI)

This interface is listed because the NGCR graphics language interface (GLI) may require kernel-level access for performance reasons. If special services are needed, then this interface will be required. It is not clear at this point whether the API will be sufficient for the needs of the GLI implementation.

INTERCHANGEABILITY

This is the characteristic of hardware and software items by which a given item can be exchanged with another without changes in the external interfaces or in any other attributes of the system in which the items are employed.

INTEROPERABILITY

This is the characteristic of systems or of hardware and software items by which each can furnish information to and accept information from the others and can use such information effectively in mutual and cooperative functions.

INTERPROCESS COMMUNICATION (IPC)

This is a means of passing arbitrary amounts of data between cooperating processes comprising an overall application on one or more processors.

KERNEL

This is defined as

1. A nucleus or core, as in the kernel of an operating system.

2. Primarily responsible for process execution and interprocess communication.

3. A nucleus or core set of primitive functions that process execution by managing a pool of resources (e.g., processor(s), memory, I/O device(s), bus interface(s)). Typically, an operating system kernel provides services for process management, memory management, time management, interrupt management, etc.

4. An operating system kernel that typically provides services for process management, memory management, time management, interrupt management, etc.

LOCAL AREA NETWORK (LAN)

This is a communication network designed for a moderate sized geographic area and characterized by moderate to high data transmission rates, low delay, and low bit error rates.

LOCAL AREA NETWORK INTERFACE (LANI)

This interface separates the LPOS software from the LAN software/hardware subsystem. The NGCR SAFENET effort may provide the interface or a set of interfaces that the LPOS software can use. If the SAFENET standards do not provide usable interfaces, then the OSSWG OSS will need to define these important interfaces.

LOCAL DEVICE INTERFACE (LDI)

The LPOS software can be written with device drivers included for the specific system devices, but a more modular system would be possible if there were standard device-driver interfaces. Device-driver interfaces make it easier to add new devices to the system or to rearrange the configuration of devices. The LDI can be at two levels: at the LPOS-to-device level if the device drivers are custom built into the LPOS, or at the LPOS-to-device-driver level if a device-driver interface is defined for the LPOS. Defining a device-driver interface allows a new device to be added to an existing OS by adding a new device driver rather than requiring a new version or modification of the OS.

LOCAL HARDWARE INTERFACE (LHWI)

This interface is usually hidden (i.e., proprietary to the LPOS vendor). It is hardware dependent and should probably not be standardized by the NGCR effort. This interface includes the interface to the backplane as a subset. While the backplane hardware itself will be built to NGCR standards, there may not be a standard, board-level, hardware interface to the backplane. This interface to the backplane and other local hardware will then depend on the design of a particular board.

LOCAL PROCESSOR OPERATING SYSTEM (LPOS)

The LPOS allocates the shared local devices among the applications competing for their resources and also supports communication and cooperation with other LPOSs at other processor nodes of the system.

LPOS-TO-LPOS INTERFACE (OSOSI)

This interface allows one LPOS to communicate with other LPOS instances in the system and allows instances to share resources and to cooperate with each other. This interface is needed if the goals of reliability and dynamic reconfiguration in a heterogeneous system are to be provided by the operating system rather than being available only if supplied by the application software. The scope of the services provided by this interface depends on the level of coordination needed among the separate processor nodes. An example of the type of communication needed can be shown when two LPOS nodes share a memory board across the backplane. The two nodes must coordinate with each other to allocate and use the memory. Without an OSOSI, the application developer will have to perform all the memory management coordination in the application. Another example of the kinds of messages needed are those to coordinate the live insertion or removal of processor boards in a running system as supported by the NGCR backplane.

MAN-MACHINE INTERFACE (MMI)

This interface is the interface between the operator or application user and the application programs. The hardware used for this interface is often some combination of special-purpose display devices and user input devices. This interface can be considered a special form of device interface for which a standard set of device-driver commands would be useful to promote software transportability and easy movement of development engineers among projects and users among embedded systems. Any standardization at this level may come out of the NGCR GLI Working Group (GLIWG). Many existing operating systems have very little support for the MMI except for (perhaps) a command-line parser.

MESSAGE

This is defined as

1. Any communication, written or oral, sent between persons; a formal, official communication, written or oral.

2. When used in the context of operating systems, LANs, and backplanes, message is a generic term that refers to either the class or any instance of a broad class of formal communications among processes, between processes and people, and among processes and controlled entities. This class is characterized by an instance of the communication having a formal protocol, specified contents, and an identified set of endpoints.

MODULE

A module can perform processing, memory functions, or server functions (i.e., I/O, security, bus interface, real-time clock). A module can itself be a processing node.

MULTIPROCESSING

This is the simultaneous and/or interleaved execution of two or more programs or sequences of instructions by a computer component, a computer, or a computer network.

MULTIPROCESSOR

This is a computer system having multiple arithmetic or logic units that can be used simultaneously.

MULTIPROGRAMMING

This is defined as

1. A mode of operation that provides for the interleaved execution of two or more computer programs by a single processor.

2. Pertaining to the concurrent execution of two or more computer programs by a computer.

MULTITASKING

This is a technique in which a program consists of two or more independent tasks that can execute concurrently.

NETWORK

This is defined as

1. An interconnected or interrelated group of nodes.

2. Two or more computers, including workstations and machines of different capabilities, possibly at remote locations, linked together by data communication channels to permit data exchange.

NETWORK OPERATING SYSTEM

This is a collection of software and associated protocols that allows a set of autonomous computers, which are interconnected by a computer network, to share resources (e.g., files).

NODE

This is a point at which one or more functional units interconnect transmission lines.

OPERATING SYSTEM

This is defined as

1. Software that controls the execution of programs. An operating system may provide services such as executive services, configuration management, fault isolation and diagnostics, memory management and allocation, communication with and management of I/O devices, common routines, debug aids, scheduling, and data management and storage. Although operating systems are predominantly software, partial or complete hardware implementations are possible.

2. A system that provides support in a single spot rather than forcing each program to be concerned with controlling hardware. It is not directly involved with the functional objectives of the real system itself.

3. A collection of system software used to control the execution of programs by managing access to critical system resources such as the CPU, memory, devices, secondary storage, etc.

4. Set of capabilities, usually in software, to extend and share the bare general purpose computing resources among one or more software applications.

PERFORMANCE LEVELS

These levels are to be determined.

PLATFORM

This is defined as one or more networks working under a common control and all with a common mission.

PROCESS

This is defined as

1. The execution of any program.

2. A collection of single threads that share resources.

3. Heavyweight process - Execution of a program or a portion of a program that includes one or more lightweight processes. Each process owns its resources and the resources are shared within the heavyweight process.

4. Lightweight process - Execution of a single thread of control in the context of a heavyweight process.

PROCESSING NODE

This is defined as one or more modules that share a common bus. A module can itself be a processing node.

PROCESSOR

This is a collection of hardware (e.g., CPU(s), coprocessor(s), accelerator(s)) that is capable of autonomously executing one or more threads of control.

PROGRAM

This is defined as

1. A schedule or plan that specifies actions to be taken.

2. The initial execution of one or more processes.

PROJECT SUPPORT ENVIRONMENT INTERFACE (PSEI)

This interface provides a means for the PSE to interact with the LPOS for loading software, testing, debugging, etc. In many systems, this interface would be removed before the system became operational. The standardization of this interface will make it easier to have a common PSE for different LPOS instances. Some systems will have a different, hardware-based, nonintrusive testing interface to the LPOS. The nonintrusive testing interface is, by its

design, invisible to the LPOS hardware and software. Therefore, it is not important to the NGCR OSIF Standard effort even if it is of extreme importance to a particular project.

REAL TIME

A real-time event or data transfer is one that must be accomplished within a time constraint, neither too early nor too late. The time constraint can be either hard (i.e., absolute with no tolerance) or soft (i.e., having a tolerance or residual value for earliness or lateness). The value of the data or accomplishment of the event is constant within the time limit.

RECORD

This is an aggregation of data items for storage/retrieval purposes.

RUNTIME ENVIRONMENT

This is the set of all capabilities provided by three basic elements: predefined subroutines, abstract data conventions, and control structure code conventions.

RUNTIME LIBRARY

This is the set of all the predefined routines in a machine executable representation that support all the functionality of an application programming language that is not supported in code generated from application programs.

RUNTIME SYSTEM

This is a set of predefined routines in a machine executable representation that is selected by the Ada compilation system from a runtime library to support functionality of the application program not supported in the generated program.

SCALABILITY

This has not been defined by OSSWG.

SOURCE APPLICATION PROGRAM INTERFACE (SAPI)

This interface is the one that is normally thought of when OS interfaces are being discussed. This is the interface (or set of interfaces) that the applications programmer uses to develop embedded systems. This is the high-

order language bindings (Ada, etc.) to the OS system calls. The BAPI is the compiled, binary version of this interface.

STRING

This is a group of related threads (of control) or "processes" that exist in multiple processors.

SUBSYSTEM

This is a group of assemblies or components or both combined to perform a single function.

SYSTEM

This is defined as

1. A set of one or more computers and the associated software, peripherals, terminals, human operators, physical processes, information transfer means, etc., that form an autonomous whole capable of performing information processing or information transfer or both.

2. An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions.

3. A group or subsystem united by some interaction or interdependence, performing many duties but functioning as a single unit.

SYSTEM SOFTWARE

This is application software designed for a specific computer system or family of computer systems to facilitate the operation or maintenance of the computer system and associated programs, e.g., operating systems, compilers, utilities.

TACTICAL SOFTWARE

This is application software that provides functions necessary to carry out the mission of the system.

TASK

This is defined as

1. A single thread of control in a single processor.

2. An Ada task - At execution time, a task object can be mapped onto a lightweight process or a heavyweight process.

THREAD OF CONTROL

This is a sequence of instructions intended to be executed at a single processor. A thread could span a distributed system (multiple processors) in some more sophisticated applications.

6.2 ACRONYMS AND ABBREVIATIONS

ANSI	American National Standards Institute
API	Application program interface
ARTE	Ada runtime environment
ARTEI	Ada runtime environment interface
BAPI	Binary application program interface
BITE	Built-in test equipment
CPU	Central processing unit
DAC	Discretionary access control
DBKI	Data base kernel interface
DBMS	Data base management system
DOD	Department of Defense
FIFO	First in, first out
FTAM	File management and access system
GLI	Graphics language interface
GLIWG	Graphics Language Interface Working Group
GRKI	Graphics kernel interface
IEEE	Institute of Electrical and Electronic Engineers
I/O	Input/output
IPC	Interprocess communications
IRAX	Intermediate resources allocation executive
ISA	Instruction set architecture
ISO	International Standards Organization
ISR	Interrupt service routine
LAN	Local area network
LANI	Local area network interface
LDI	Local device interface
LHWI	Local hardware interface
LPOS	Local processor operating system
LSCI	LPSO-SRAX coordination interface
MAC	Mandatory access control
MMI	Man-machine interface
NDI	Nondevelopmental item
NDS	Nondevelopmental software
NGCR	Next-generation computer resources
NOSC	Naval Ocean System Center
NSWC	Naval Surface Warfare Center
NUSC	Naval Underwater System Center
OCD	Operational concept document
OPNAV	Chief of Naval Operations

OS	Operating system
OSA	Open-system architecture
OSI	Open systems interconnect
OSIF	Operating system interface
OSOSI	LPOS to LPOS interface
OSS	Operating systems standards
OSSWG	Operating Systems Standards Working Group
PSE	Project support environment
PSEI	Project support environment interface
RAX	Resource allocation executive
RDT&E	Research, development, test, and evaluation
ROM	Read-only memory
RPC	Remote procedure call
RT	Real time
RTNI	Real-time non-intrusive testing
SAPI	Source application program interface
SPAWAR	Space and Naval Warfare Systems Command
SRAX	System resource allocation executive
SRAX-C	SRAX-centralized part
SRAX-L	SRAX-local part
TCB	Trusted computing base
TCP/IP	Transmission control protocol/internet protocol
TCSEC	Trusted Computer Systems Evaluation Criteria
USAP	User service access point

APPENDIX

REFERENCE MODEL FOR EMBEDDED OPERATING SYSTEMS

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

10.1 INTRODUCTION

The OSSWG reference model is a conceptual model that provides a context for application program developer's requirements, for comparing existing operating systems, and for standards specification (reference 4). It provides a small, common set of conceptual, embedded-system building blocks with associated interfaces and functionality (reference 7). Many of these system building blocks will be the results of other parts of the NGCR project.

Consider the international standards organization (ISO) open-systems interconnect (OSI) reference model as an example of a reference model. That model defines seven protocol layers and associates each network communication function with one and only one layer. The model does not, however, specify a given protocol or protocol implementation for a given layer.

This OSSWG reference model is a model with the full embedded system as its scope, including some aspects of the project support environment (PSE). The model provides the basis for defining a set of concepts and conventions used by the system developer in designing and implementing the control system for the computer resources employed in an embedded system. Objectives of this model include describing the requirements for the portability and reusability of software components of the system and for the interoperability of software and hardware components.

Many operating systems are built on a set of abstractions that make certain aspects of an application's execution "invisible" to the application itself. While the higher level view of the system can be valuable to the application programmer, this reference model will need to make some of those aspects visible so that the functionality of the operating system can be discussed and evaluated. Therefore, the discussion of an operating system feature or aspect in the model does not necessarily imply that the feature should be easily or directly accessible (i.e., "visible") to an application program.

The operating environment of operating systems built to the NGCR OSIF Standard will differ greatly depending on the size and requirements of the system and its intended mission. It is expected that systems using an OS compliant with the NGCR OSIF Standard will not use all the features discussed here or specified by OSSWG OSIF requirements documents, but will use tailored subsets for each particular application system.

A reference model must satisfy conflicting requirements similar to those encountered in more traditional modeling disciplines. The model must be structured enough to encourage the generation and use of standards and standard components. Yet it must be flexible enough to accommodate tailored and special-purpose components necessary to meet real-world needs. The reference model should satisfy the following requirements:

- Be simple.
- Accommodate existing and imminent embedded systems standards, both hardware and software.

- Allow incorporation of both standards-based and proprietary subsystems.
- Reflect the full scope of the application program developer's functional requirements.
- Allow system scaling.
- Accommodate new embedded system technology.
- Provide a means for comparing existing operating systems.
- Provide direction for future standardization and integration efforts.

It should be noted that the definition of this model is an engineering task and not a scientific one. There are many possible models, and while it might be interesting to contemplate an optimal one, an adequate solution is all that is required. In addition, it should be noted that this model is intended to be conventional within computer science. The intention is not to break new ground, but to establish simple terminology and concepts for identification and resolution of architectural issues. It should also be noted that some of the text for this appendix was taken from the IEEE POSIX System Architecture, Standard 1003 (reference 8).

10.1.1 *Multiple System Views*

The OSSWG reference model is expressed by using multiple views of embedded systems and their operating systems' interfaces. No one viewpoint seems sufficient to concisely describe the needs and goals of the NGCR OSSWG program.

In section 10.2, a model is developed that provides an introduction and overview of the system considered. The overview model defines system elements, to expose interfaces across which service requirements should be satisfied. The elements are chosen to expose those interfaces that are significant to the embedded system's developer.

Section 10.3 describes the important or critical interfaces between the embedded operating system and entities external to the operating system.

Section 10.4 describes the basic services available across the interfaces of the operating system. The services are defined in a generic way, based on the model and current industry practice.

Section 10.5 discusses various application domains. This is a brief description of how the target systems differ with respect to several important requirements. The intent is to illustrate and bound the large variations that are expected among systems that may use the operating systems standards.

10.1.2 *Abstraction and Levels of Aggregation*

For purposes of understanding system concepts, the software engineering principal of information hiding is exemplified by representing the end system as a single machine. (See figure 10-1 for a context diagram of the virtual machine and its environment. Once the total system concept is understood, the context diagram is subdivided to reveal some of the lower detail.

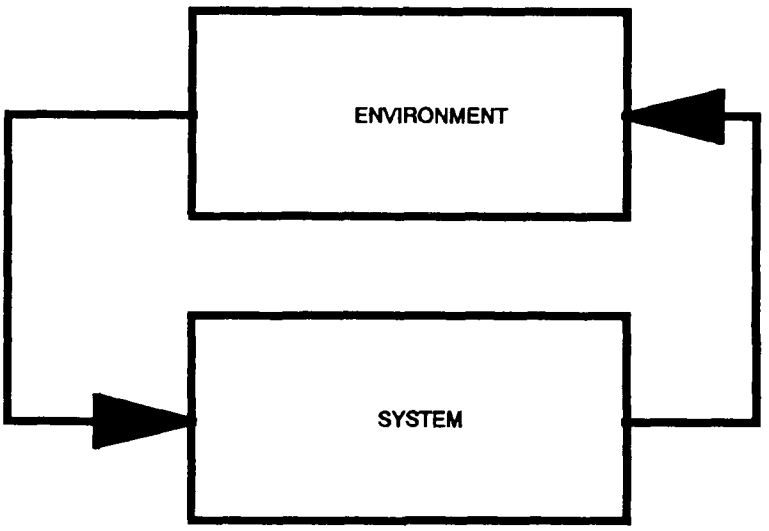

Figure 10-1. System Context Diagram

10.1.2.1 *User Roles.* The model must be expressed in multiple viewpoints, because a number of different users with various roles must use the OSIF standards. Some of these roles are application developer, application user, target system operator, PSE user, and PSE system operator. This list of roles is not an exhaustive list but, for purposes of the model description, this list will suffice.

The application user is the end user of the system who may have no knowledge of the operating system interfaces used to provide the functionality seen when the system is used.

The system operator, if there is one for a particular system, has special privileges and utilities to modify the system's data base and configuration to meet changing needs of a structure of the particular application.

10.1.2.2 *Model Description.* The model, as described from the point of view of the application developer role, is used to describe the reference model so that:

1. Application developers will have the proper services to meet their requirements.

2. Vendor implementation will not be constrained unnecessarily.

The reference model is represented by several levels of abstraction, with different objects being more significant at each level. The five levels of abstraction are as follows:

1. System design-level focus - the integration of multiple application programs into a cohesive system.

2. Program design-level focus - the integration of application modules and system services.

3. Operating system-level focus - the organization of and interfaces to the system services provided by the operating system. This is the SAPI level.

4. Logical device-level focus - the logical devices from which the system is composed.

5. Physical device-level focus - the physical devices from which the system is composed.

Figure 10-2 shows the end user virtual machine internal structure from the system designer's point of view. The major objects are applications and the systems devices. This view does not indicate the system hardware. The system might be implemented by one hardware processor or by a large number of processors. The hardware is assumed to be a single virtual processor. At this level of abstraction, the system is composed of one or more application programs that interact with each other, with the devices in the system, and with the users by use of the MMI, if appropriate.

Figure 10-3 shows a diagram of an application from the view of the program design role. The internal boxes, each labeled "service group," represent modules providing system services. Those modules may be loaded with the application code or may be part of the shared operating system code. At this level, the application designer sees the application as composed of application modules and OS modules that interact to perform the activities of the application.

The SAPI describes the interface between the aplication and the operating system from the source code level. The SAPI defines the program designer's means to access the functions and objects of the operating system. The system's devices and other software entities are seen as being available to an application program through the services of the OS.

The logical device defines the interfaces to objects of generic types attached or accessible to the system. The particular characteristics of specific hardware devices are hidden by the device driver and the operating system.

The physical device interface is used by the writer of the device driver and by the application developer if the device needs to be accessed at a low, physical level for some special application.

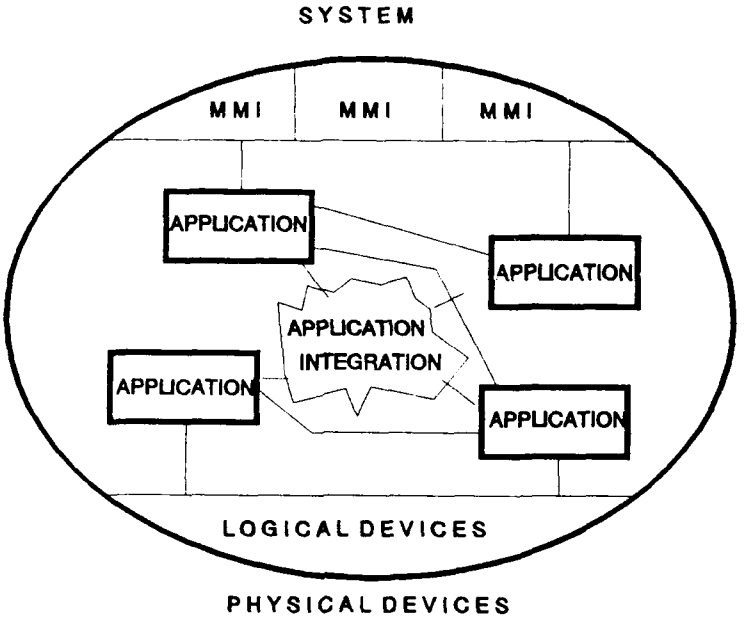

Figure 10–2. System Designer's View of the System

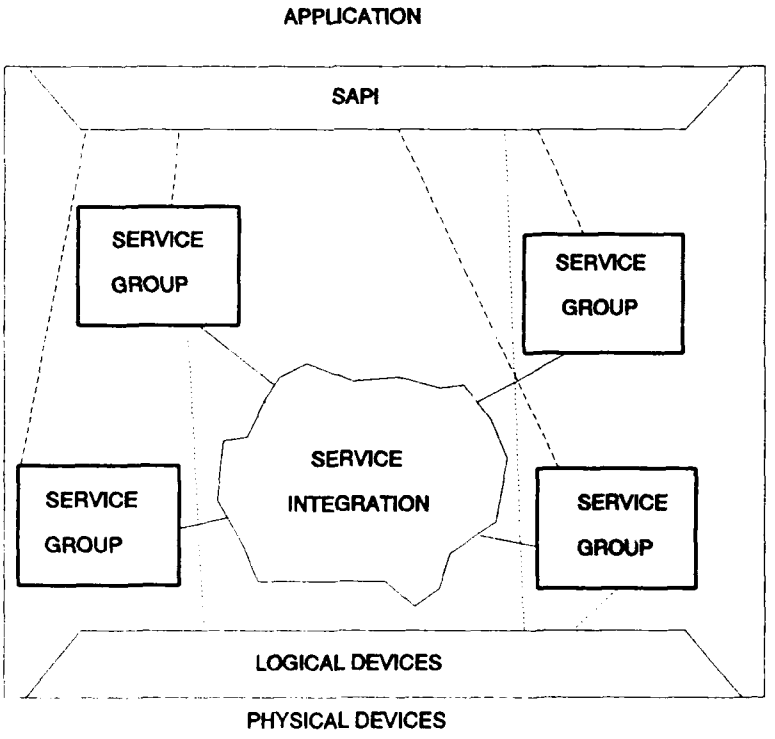

Figure 10–3. Application Programmer's View of the System

10.2 SYSTEM OVERVIEW MODEL

Figure 10-4 represents the embedded system as viewed by a system developer. This view corresponds to the program design level of abstraction. External features visible to the application developer include a variety of devices used to display and enter data. These devices include those used to display and enter data. Also included are sensors and effectors that provide a means for the system to interact with the real world.

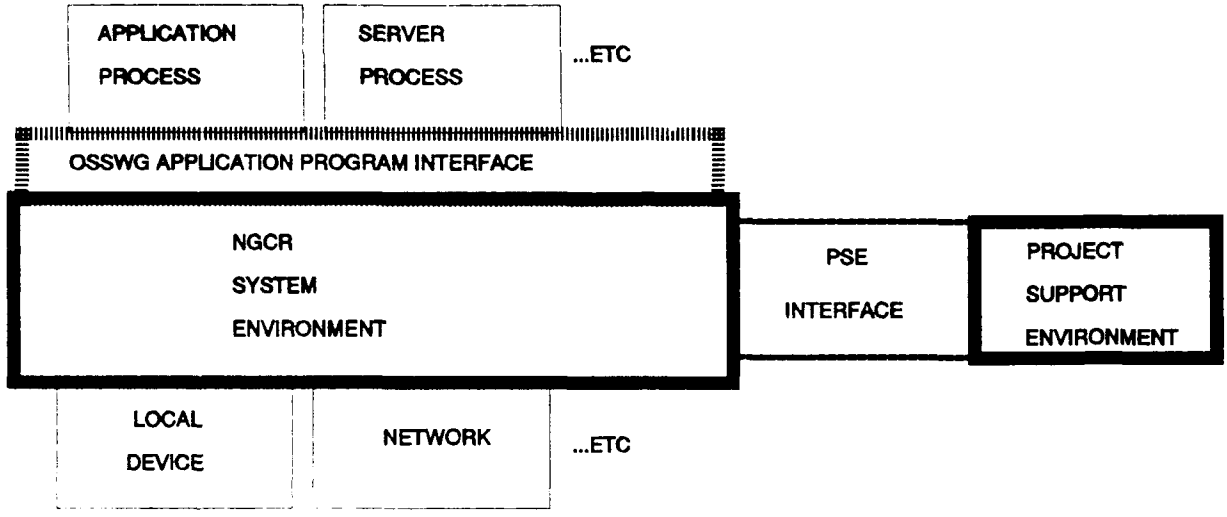

Figure 10–4. LPOS Node Model

All runtime features can be conceptualized as being contained within the NGCR system block. All of the operating system features may be available locally or remotely. Some of the OS functions may be performed remotely if the system is a distributed system with multiple nodes. The term "node" is used to describe a single hardware subsystem that executes at any particular time, a single thread of operating system control. A node may actually be constructed with multiple hardware processors, linked for self-checking or redundancy or both, or may be a processor with auxiliary processors (such as floating point or I/O processors) attached. The aspect of the processor node view of hardware resources that is singular is that it contains one and only one local operating system.

Two constraints are defined for the basis for the system node overview model as follows:

1. The application software system is represented as the execution of a collection of processes, where a process executes on a single processor node and contains a single, schedulable thread of control as seen by the local operating system.

2. Some mechanism for communications among these processes exists, whether communicating processes are located on the same or different nodes.

The following parts of this section describe the basic elements of the distributed NGCR system and the relationships among them. This discussion defines the paradigm for the descriptions of services, interfaces, and target domains that follow in sections 10.3 through 10.5.

10.2.1 *Single Node/Application Model*

Figure 10-5 identifies the major elements of a local processor node that are important to the embedded system developer and the relationships among them. While the LPOS is shown as a single block, its implementation is undefined by this model and it could be structured in many ways. For example, it may very well consist of a proprietary base OS, not compliant with the NGCR OSIF standards, with specific NGCR OS services and interfaces implemented "on top of" that proprietary OS.

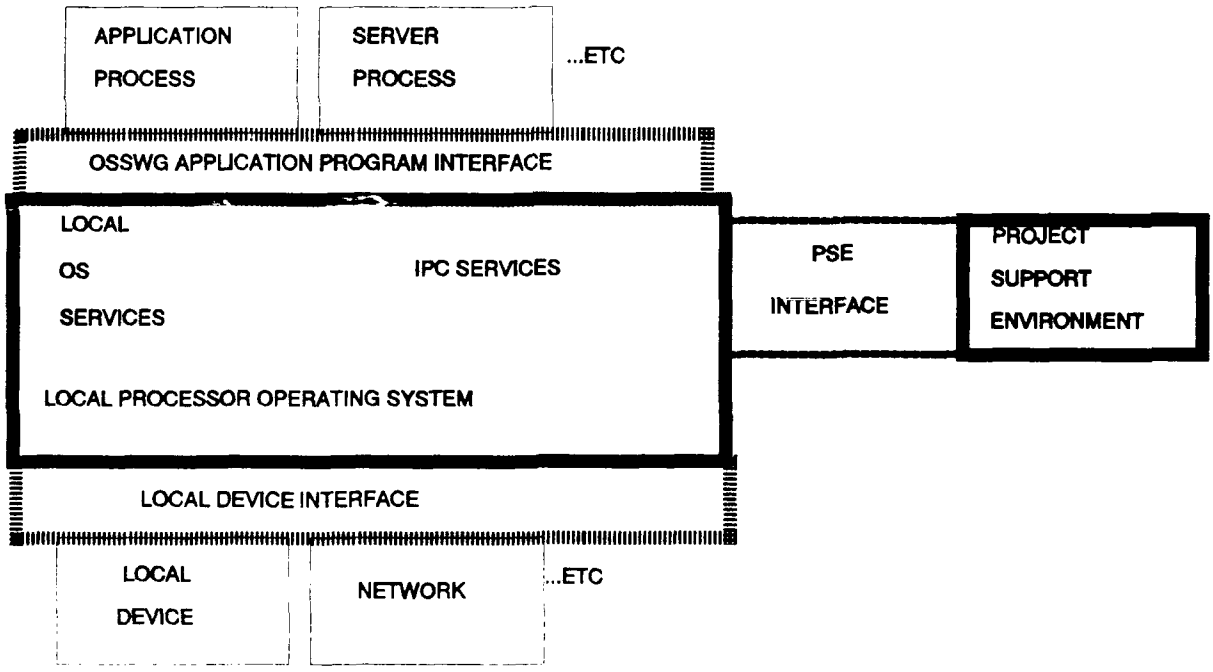

Figure 10-5. LPOS from Operating System Perspective

The elements of the node include the following:

- Application programs
- LPOS
- API
- Local devices
- PSE.

One or more application processes may run on the processor simultaneously, as represented by the process rectangles at the top of the figure. The applications run as independent software entities and communicate among themselves by means of a variety of communications mechanisms provided or managed by the LPOS.

The applications make use of devices attached to the local processor to perform a wide variety of actions. These local devices are represented along the bottom edge of the figure, and they include sensors, effectors, and direct or network connections to other computing systems.

The LPOS allocates the shared local devices among the applications competing for these resources. Processor time, memory, and other finite processor resources are also shared among the applications and mediated by the operating system. In addition, the LPOS supports communication and cooperation with other LPOSs at other processor nodes of the system.

The block labeled LPOS in the figure actually contains only the runtime elements of the operating system that usually run in supervisor mode or protected mode. These elements of the LPOS are the parts that handle system service requests from the application programs. Other parts of the LPOS may run as server processes or as library routines linked with application programs. These server processes may have special authorizations or capabilities but are scheduled and serviced by the runtime elements of the LPOS in the same manner as user processes.

The LPOS may also support an ARTE. The SAPI, the compiler vendor supplied compilation library, and the interface code generated by the compiler together form the complete ARTE. The ARTE functions may be implemented using services of the OS, in code modules from the Ada library, or code generated in place by the Ada compiler. While it is expected that the OS will support the needs of programs written in Ada, other languages and their runtime support will also be needed for particular projects.

For the OS to protect system integrity and ensure system data base consistency, applications competing for system resources must access all system resources using system service requests. The formal definition of these requests (or system calls) defines an API. The API will specify a sufficient interface between the application program and the underlying operating system and includes the operating system services that are described in section 10.4.

The API has several different representations. One set of representations is illustrated by the SAPI, which is a programming language binding to the API for some particular source language. For each language allowing service requests to the operating system, there is a SAPI, a set of subprogram calls to be invoked to access operating system services. This is the representation used by the programmer and is a primary interface used in the PSE. Another representation is the BAPI, which is the calling mechanism used by the compiled code to access the operating system routines that are not part of the application code. The name "binary application program interfaces" is meant to imply that this interface is at the machine code

level. The BAPI will, in most cases, be LPOS implementation and processor dependent, but might be standardized for a single processor type or a family of processors to allow some degree of portability of compiled code across different implementations of the LPOS.

The project support environment interface (PSEI) is the block on the diagram between the LPOS and the PSE. This interface allows communication between the PSE and the LPOS in a development environment. This interface may not be subject to standardization at this time, depending on an analysis of the risks and benefits of such a standard PSEI. The PSEI provides the following services:

1. Down-loading of compiled programs.

2. Up-loading of execution-time debugging information.

3. Remote control of the embedded system by the user of the PSE, including the execution of debuggers running partially on the PSE and partially on the target system.

Note that these kinds of operations are similar to those required for coordination of multiple-user programs running on multiple, distributed processors except that very detailed knowledge of the execution environment on the target will need to be communicated back to the PSE. Most of the functionality of this interface may be available from the SAPI/BAPI.

The LDI block in the figure is the set of device drivers used by the LPOS to access the different devices. The interface between the device drivers and the LPOS is the LDI.

The primary interface for OSIF standardization is the SAPI. With a standard SAPI, an application routine can be transported to a new target system by recompiling and relinking the source code with the new library. A standard SAPI will also support interoperability and software reuse at both the subprogram, subsystem, and system level.

A development version of the embedded system may have an RTNI testing device attached to it for monitoring the system's performance and debug the application (and perhaps the system) software. An RTNI device is not shown in the model because a truly non-intrusive device will not be "visible" to the software or to the operating system of the tested system. An RTNI device may, under some conditions or modes (e.g., during test setup), be visible to the system as a special-purpose device and an NGCR OSIF Standard compliant operating system should be capable of communication with such a "visible" device. The RTNI system itself may contain a computer system that uses an NGCR OSIF Standard compliant operating system to run applications that collect data about the tested system.

10.2.2 *Network/Application Model*

The system will now be expanded to expose network-related interfaces. Setting aside the local node model for the moment, figure 10-6 relates the application processes to a conceptual model based on the OSI reference model for network services. The figure shows an implementation where the networking support and operating system are integrated; other implementations are possible depending upon the communication services provided by the OSIF standard.

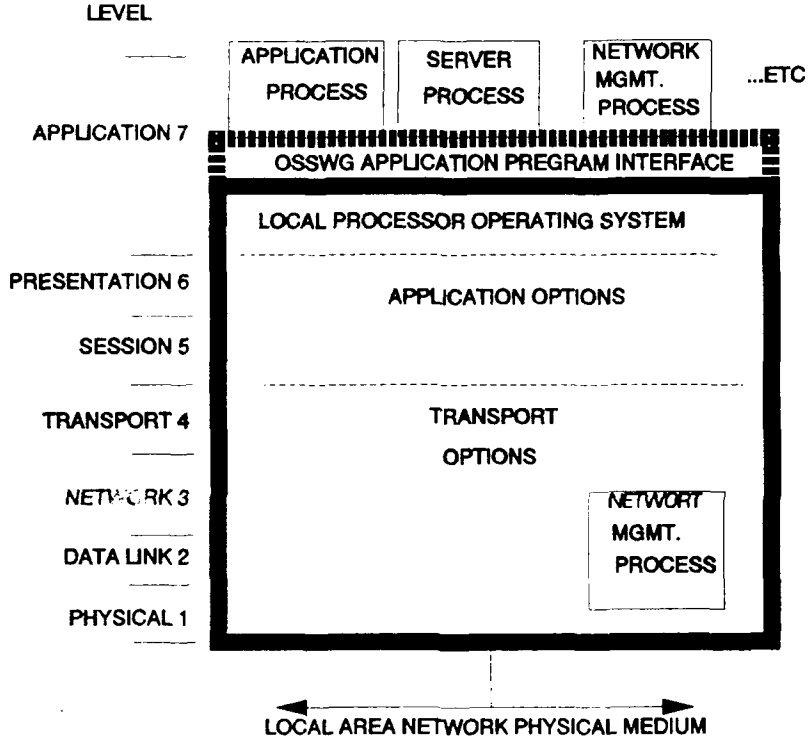

Figure 10-6. Network Model from Applications Perspective

Applications gain direct access to the network services at levels 4 through 6 by use of service requests specified in the API. This API may be part of the operating system's API or may be a separate API provided by the networking system. In the case where the network services are made available to users by OS interfaces, the OS treats the network as just another system resource to be allocated among the competing processes. Connection-oriented, connectionless, and multicast data transport services may be available.

API data transport services are usually process to process. This means that a message is delivered when it is presented to the destination process, not just when it enters the destination processor node. In addition to process-to-process services, there may be a need for other transport services such as some form of broadcast or mailbox services. There may also be a need for transport services that are location or processor dependent.

Note that communications methods other than network services may be available to processes located on the same node or closely coupled nodes (e.g., shared memory, event flags). These options may not be available, however, for processes located on nodes that do not share a very high bandwidth communication medium.

In cases where reconfigurability is important, it may be useful to use a network communication interface between processes on the same local processor. For example, it is possible that one of the communicating processes could potentially be moved to a different processor node. If the processes will always be located on the same node, other methods may improve performance. This is an application-level architectural decision that has substantial impact on design and implementation of distributed system applications. The actual source level interface for local and remote communication between processes may be the same with the responsibility falling to the OSs and/or compilation systems to determine the location of a process and facilitate the proper communications.

10.2.3 *Network Communication Model*

Figure 10-7 integrates the node/application and network/application models into an element of a local area network coupled distributed system. No new elements are introduced in this integrated model. A major feature of the model is the integration of the network protocols with the operating system on the local node. Note that the upper network protocol layers are closely associated with the operating system. This is driven by the fact that the session layers provide communications services among processes. The process is an operating system construct and is managed by the operating system, while the session and datagram services are network constructs and are managed by the protocol software. This may require close coordination and integration between these software elements, and can be a major source of difficulty during development, integration, and operations. It can be a serious risk element, especially if the operating system and network protocol software are procured separately.

Note that the application process may pass service requests to the operating system by the API to gain access to network services. As discussed above, the API provides data communications transparency to the applications. This means that the complexity of the network is hidden from the applications behind the API.

Network management functions may be associated with any network layer. Figure 10-7 shows network management processes associated with the upper protocol layers, as well as with the lower layers. This is because the upper layers are closely associated with the operating system and may use operating system services to perform network management functions. The lower layers, however, are more closely associated with the physical media and may not have direct access to the processor. In any particular system, the network management may only be implemented in one of the two places or may even be implemented in a separate processor with its own OS.

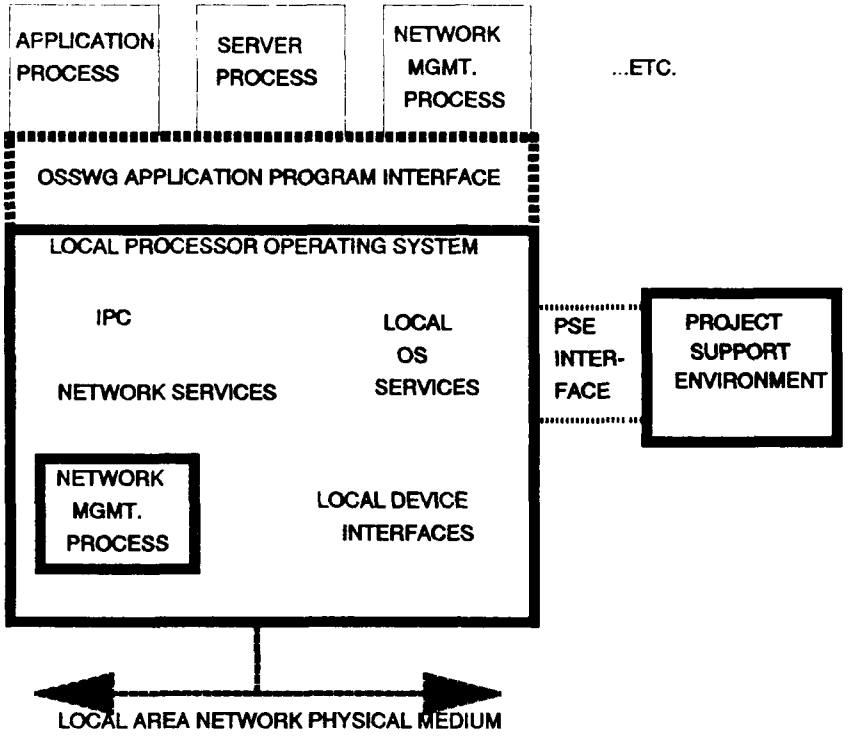

Figure 10-7. Integrated System Node

Often the operating system allows the application programmer to refer to entities attached to the local network by logical names rather than by network address. That capability can be very useful for separating the specification of the location of an entity from the code that uses it; however, some applications will need to refer to network entities by their physical address or at least be able to determine the location of an entity.

10.2.4 *Program Distribution*

Many Navy computer languages do not have any support for concurrency but rather depend on the facilities provided by the operating system by use of calls to OS services. The Ada language is an exception because it has language-level support for concurrency; other exceptions are languages designed for signal processing. With respect to programs written in languages that support concurrency, there are two levels of concurrency. Either level (or both) may be mapped to OS processes. The first is Ada task level concurrency where each unit corresponds to either an Ada main subprogram or to an Ada task. Concurrency at this level considers the relative priorities and scheduling of Ada tasks within a single program and their communication by means of language constructs. The second level of concurrency is program level concurrenc. A unit at this level is a single (Ada, etc.) program together with all of its dependent tasks. Concurrency at this level considers the relative importance of the individual programs in the

system competing for system resources. Also, program-to-program communication is by the operating system or shared data because the Ada language provides no communication facilities at this level outside of file I/O.

When an application is to be distributed across multiple processor nodes that may not share common memory, there are several ways to partition Ada programs. Two of the partitioning methods correspond to the levels of concurrency above. A program can be distributed at the Ada task level with the OS, the compilation system, or the developer deciding where each task is to execute. Distribution at this level would require communication across processor nodes with full Ada tasking semantics. Distribution at the program level means that a program and all of its tasks execute on a single processor node. If distributed processing is needed, then the application developer must divide the application into separate programs that communicate by calls to the communication facilities of the OS. This is current practice in most real-time systems including Ada-based systems.

Distribution can also be at the "virtual node" level. A virtual node consists of a collection of related Ada library units. A program may be one virtual node, which corresponds to the program level distribution, or a program may be composed of several virtual nodes, which together contain all the library units of the program. A virtual node is assigned by the developer or the OS to a particular processor node. More than one virtual node of a program can be executing on the same processor node, but a virtual node may execute on only one processor node at one time.

10.2.5 *System Resource Allocation Executive*

Conceptually, the *system resource allocation executive* (SRAX) is the single operating system for the whole system if one exists; it manages system resources across processors so that to the applications developer, the system, or some aspects of it, appear to be controlled by a single centralized operating system. The SRAX is primarily responsible for scheduling and allocating resources that affect more than one local processor node. But, because most resources are local to some local processor node, the SRAX must cooperate with or control the local scheduling mechanisms. Note that the coordination across local processor nodes includes nodes that have different types of processors. The communication between the different LPOSs require a set of functions and protocols that may not be part of the API.

Below the level of system-wide coordination and management, there may be several intermediate resource allocation executives (IRAXes). An IRAX manages a particular resource or set of resources for a set of processor nodes. The organization of the IRAXes and the SRAX may be strictly hierarchical with the SRAX at the root node of the tree structure, the IRAXes at intermediate nodes, and the LPOSs at the leaves of the tree.

Another system organization could have IRAXes assigned particular resources for management within a small number of LPOSs. With this organization, a single LPOS would interface with multiple IRAXes for different

resources. These lower level IRAXes could then communicate with a higher level IRAX for more global management of that resource type.

A platform may have several SRAX-level clusters of local processor nodes that share a communication network but do not cooperate with each other for resource sharing at the operating system level. This would effectively provide multiple SRAX systems that can communicate across a network. Also on the network may be simple, independent systems that may not need any interprocessor coordination and only have the LPOS part of an operating system with no SRAX-level OS.

Figure 10-8 shows two nodes of a multinode system. Many embedded applications require that the nodes of the system be able to coordinate services and resources. This can be done all at the application level with the application programs communicating with each other and then with their individual LPOS, which will be fully supported by the API; such an application system would have no need for SRAX or IRAX level services. The LPOSes could also communicate and coordinate resource usage directly by use of the SRAX, which may provide such services as dynamic load leveling and automatic reconfiguration.

There are many ways that the RAX (i.e., the SRAX and the IRAXes) could be implemented. They could be implemented as a single, centralized program running on individual processors. They could be implemented in a fully distributed manner such that every local processor node has its own portion of a RAX program. Parts of a RAX could even be implemented in hardware for high performance. Realistic implementations would probably have some services centralized and others distributed on some or all of the local processor nodes. Even with a centralized implementation, there may be one or more "backup" processors that could take over execution of the centralized part of a RAX if the initial one fails.

Figure 10-9 is a diagram of the different parts of the operating system and their location for one very simple implementation scheme. The SRAX in the figure has two components: the centralized part (SRAX-C) and the local part (SRAX-L). The centralized part executes on one processor and coordinates the other processors, while the local part of the SRAX and the individual LPOS parts execute individually on each processor. A system with three processors has little need for multiple levels of resource management so there is no IRAX shown in the figure.

The different parts of the SRAX communicate with each other to schedule the resources of the system. It is this communication that makes system-level control possible but also can potentially cause a severe communication load on the system.

Figure 10-10 is an example of one possible way a system could be organized. This figure shows the logical connections between the different levels of resource management; the actual communications may be by means of a single linear or ring network or by means of gateways on various network segments. The example has two levels of IRAX resource management and, in this system, a single LPOS may have two or three IRAXes that manage different resources that the LPOS uses.

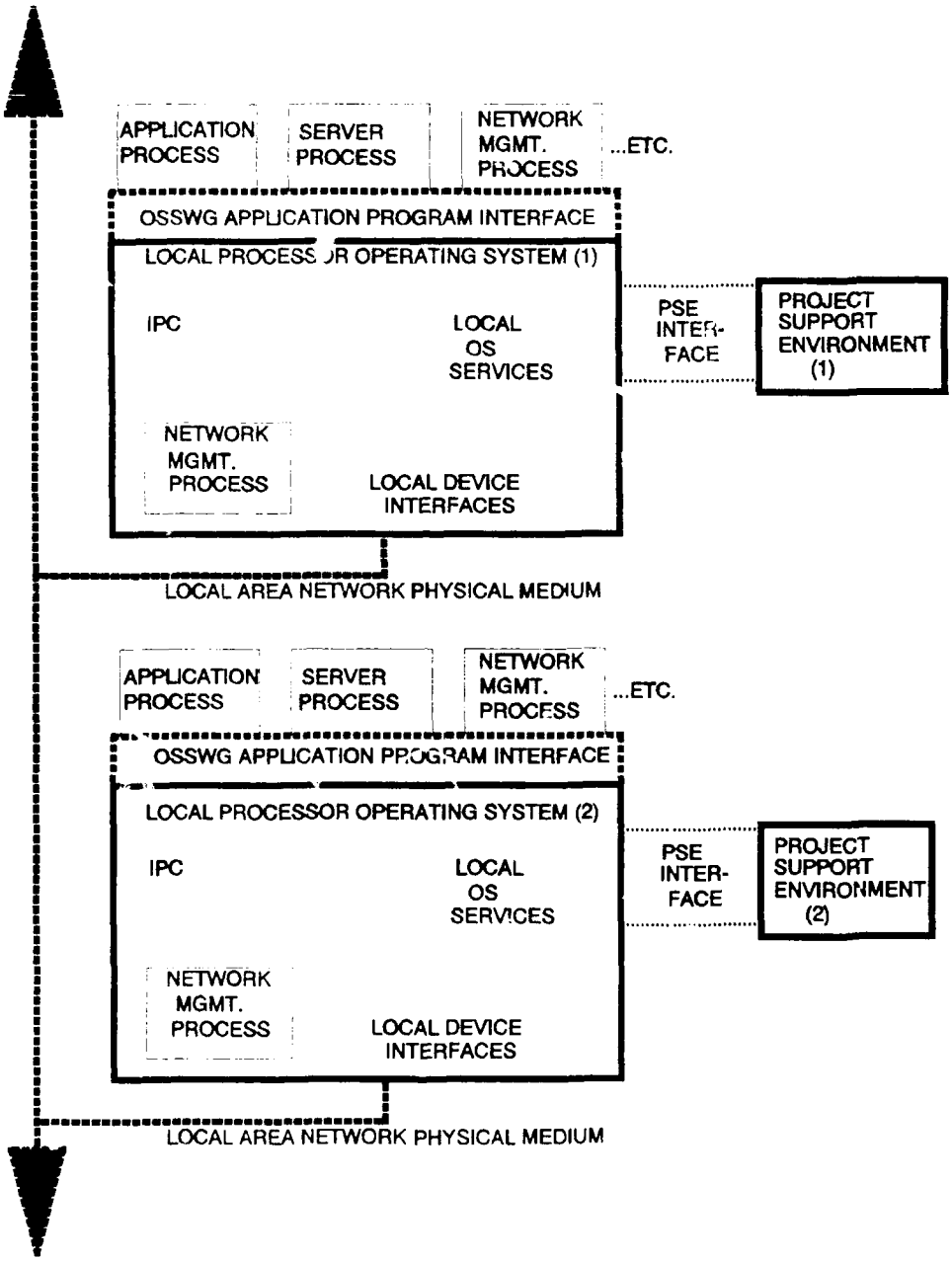

Figure 10–8. Distributed System Nodes

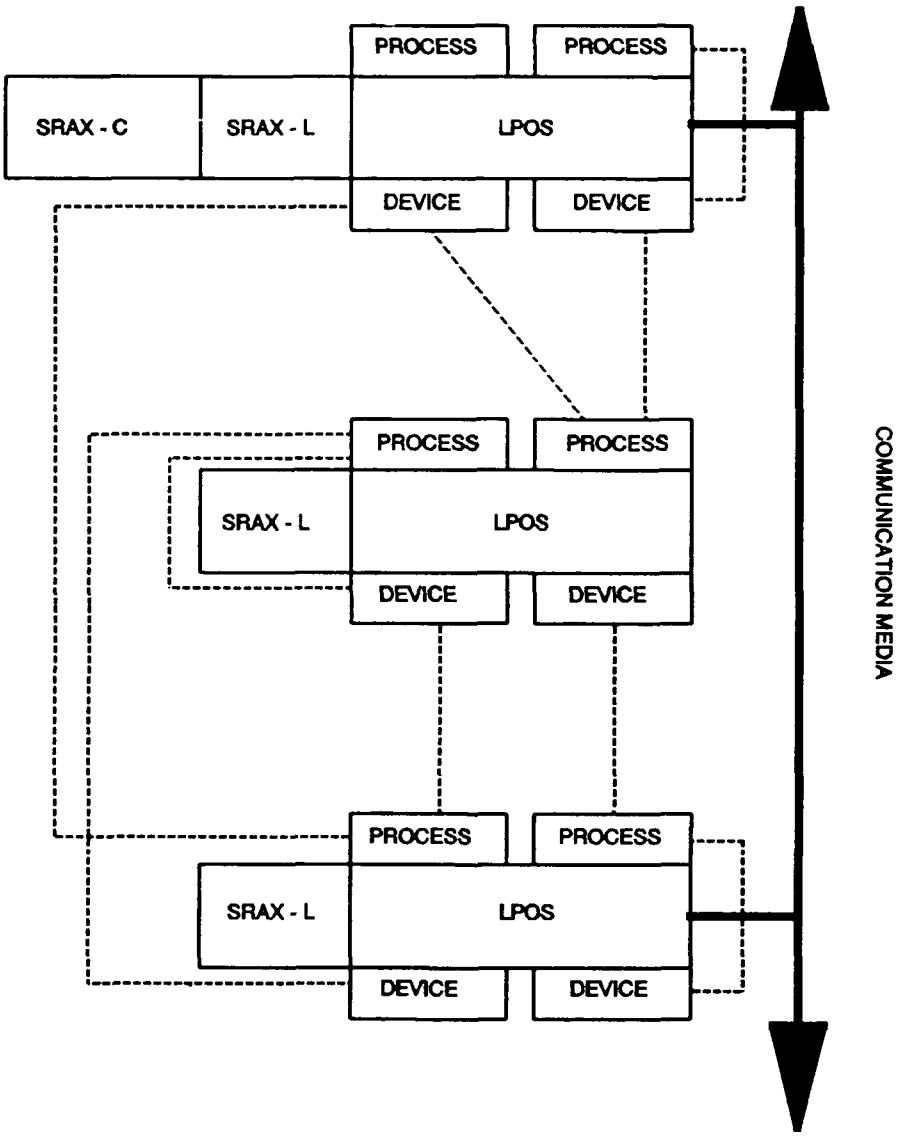

Figure 10–9. LPOS and SRAX Parts of the Operating System

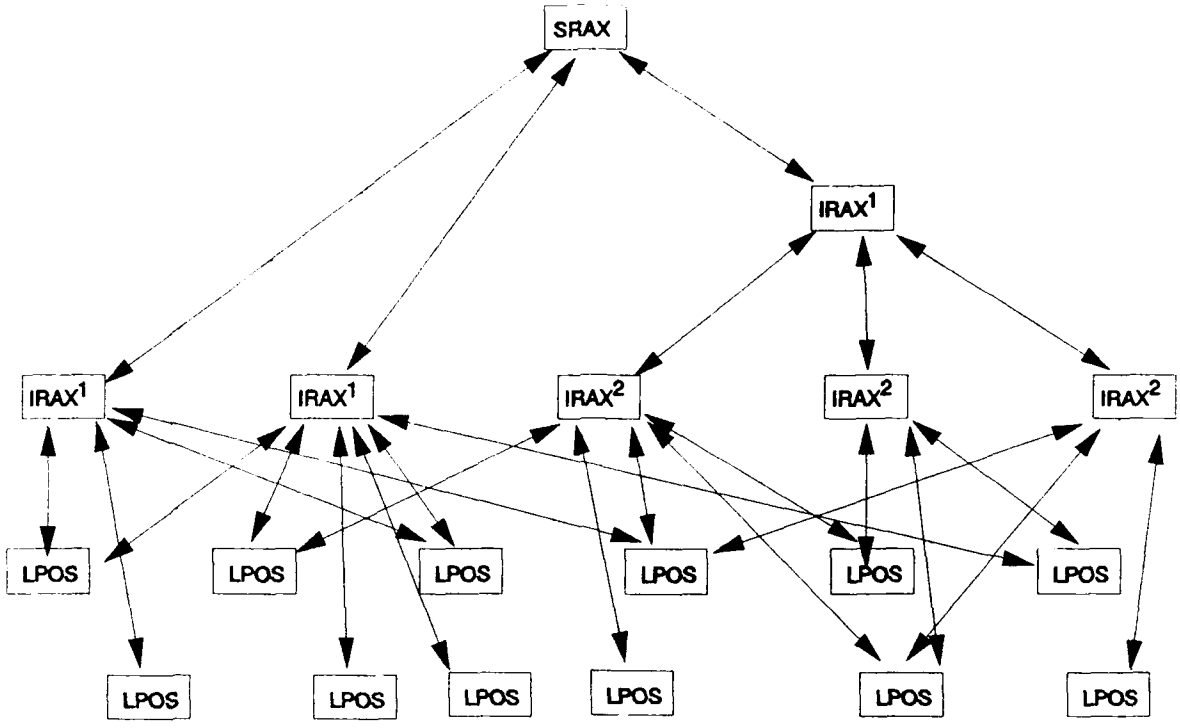

Figure 10-10. Example of System Organization with SRAX and IRAX

10.3 CRITICAL INTERFACES VIEW

This section discusses operating system interfaces that are necessary to meet the goals of the NGCR project. These interfaces are defined by the entities that the OS and application programs must interact with (see figure 10-11). These interfaces arise from the need for the different components to be integrated into a cooperating system. Not all of the interfaces are visible to the applications developer and not all need to be standardized, but all of the interfaces will be present in one or more implementations. The interfaces may be provided by compiler or operating system vendors or by the application programmer rather than being part of the NGCR OSIF Standard or any other NGCR standards.

The viewpoint taken is that of the LPOS, i.e., the component of the total operating system that executes on a local processor node. An LPOS may communicate with other LPOS instances by use of the backplane, an LAN, and shared memory. The following interfaces are discussed:

- ARTEI
- BAPI
- DBKI
- GRKI
- LANI
- LDI

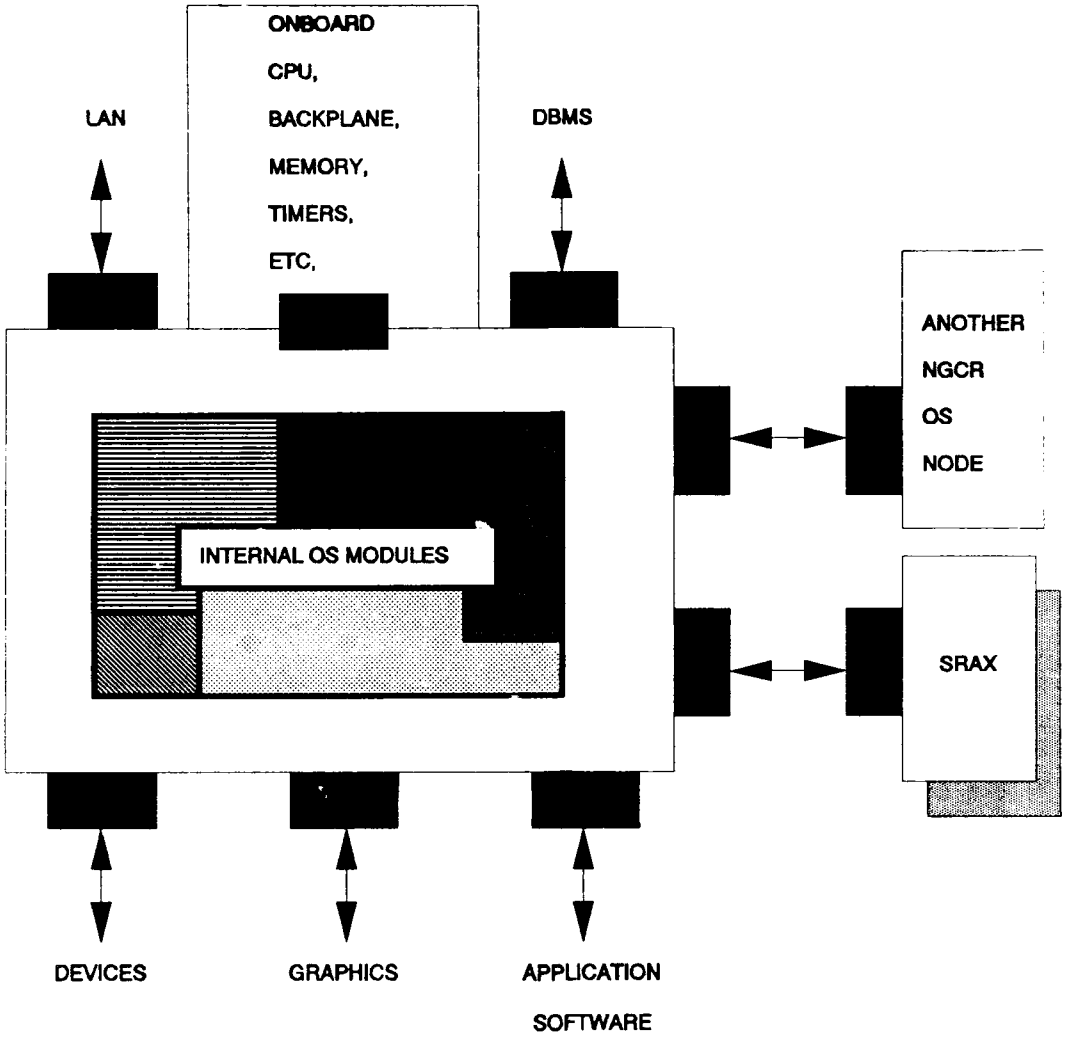

Figure 10-11. Entities that Interface with the Operating System

- LHWI
- LSCI
- OSOSI
- PSEI
- SAPI
- MMI.

10.3.1 *ARTEI*

The interface between the application Ada programs and the runtime environment required by the Ada programming language is

Ada runtime environment ‹-› application software.

Figure 10-12 shows that the ARTE may be implemented in three parts: (1) a part loaded with each application program from the Ada runtime library provided by the compiler vendor, (2) a part that is generated by the Ada compiler during the translation of the Ada program, and (3) a part that is a subset of the OS interfaces. It is an issue as to how much of the necessary ARTE is part of the operating system and how much should be provided by the compilation system of a compiler vendor. Most likely, some parts of this interface will be provided to allow applications to modify or "tune" the runtime system and other parts of the interface will only be used by compiler generated code. Standardization of the interface to the OS part of the Ada services will allow the efficient integration of the Ada runtime environment and the LPOS. This interface is one aspect of the more general high-order language binding interfaces, which may include language bindings and runtime support for such Navy languages as CMS-2 and Lisp.

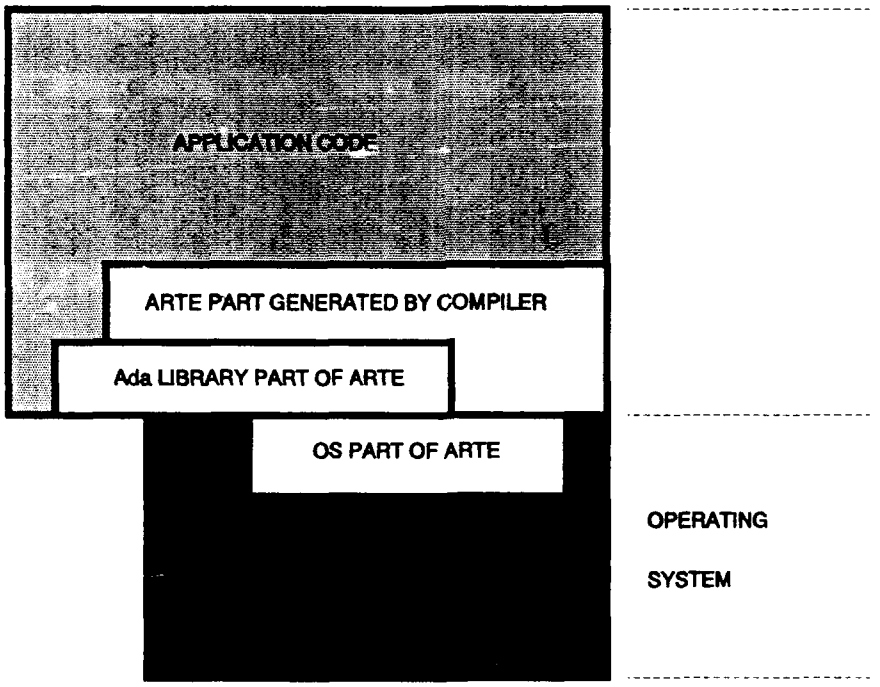

Figure 10-12. Parts of the ARTE

10.3.2 *BAPI*

The interface of the machine code level calling sequences to the LPOS is

Binary application software <-> LPOS.

It is the binary level version of the SAPI and is sometimes referred to as the BAPI. This is logically the same interface as the SAPI that will be described

but may require a separate standard if the compiler vendors are to be decoupled from the LPOS vendors. As an illustration of a BAPI, some machines use a set of software interrupt instructions to call operating system routines; the specific interrupt numbers and the specific conventions for parameter passing and stack management are part of the BAPI. Without a common interface at the binary level, a separate version of each compiler (or at least its code generator) may be required for each different implementation of the LPOS software, and code compiled by different compilers may not operate together. Note that even with standardization of the BAPI, there is, at best, a set of standards, one for each processor or processor family.

10.3.3 *DBKI*

This interface is

Data base kernel <-> DGMS.

Those systems that will require a multilevel, secure OS will require that some low-level parts of system-level components be fully integrated (i.e., be part of the TCB); therefore, the DBMS must have efficient access to the TCB. If the TCB is integrated with low-level parts of the OS, then those OS components must provide the needed interfaces for the DBMS The DBKI is the interface that the DBMS system uses to access the TCB or specialized OS components. Without a DBKI built into the LPOS, a DBMS may have to be built "on top of" the OS, which could lead to poor DBMS performance. If the data base kernel interface is of general usefulness to application programs, then this interface may simply be a subset of the application program interface (SAPI and BAPI) to the operating system and therefore not a separate interface. It could also be a separate, special-purpose interface, which is provided only for those systems that use a DBMS.

10.3.4 *GRKI*

This interface

Graphics language <-> LPOS

is listed because the NGCR GLI may require specialized, low-level OS functions for performance reasons. If special services are needed, then this interface will be required. It is not clear at this point whether the API will be sufficient for the needs of the GLI implementation.

10.3.5 *LANI*

This interface

LAN <-> LPOS

separates the LPOS software from the LAN software/hardware subsystem. The NGCR SAFENET effort may provide the interface or a set of interfaces that the LPOS software can use. If the SAFENET standards do not provide usable interfaces or if an integrated set of communication interfaces is desired, then the OSSWG OSIF Standard will need to define these important interfaces.

10.3.6 *LDI*

This interface is

Local devices <-> LPOS.

The LPOS software can be written with device drivers included for the specific system devices, but a more modular system would be possible if there were standard device-driver interfaces. Device-driver interfaces make it easier to add new devices to the system or to rearrange the configuration of devices. The LDI can be at two levels: at the LPOS to device level if the device drivers are custom built into the LPOS or at the LPOS to device driver level if a device driver interface is defined for the LPOS. Defining a device driver interface allows a new device to be added to an existing OS by adding a new device driver rather than requiring a new version or modification of the OS.

10.3.7 *LHWI*

This interface

Local hardware <-> LPOS

is usually hidden (i.e., proprietary to the hardware vendor). This interface is hardware dependent and should probably not be standardized by the NGCR effort because it is only useful when processor boards are standardized. This interface includes as a subset the interface to the backplane. While the backplane hardware itself will be built to NGCR standards, there may not be a standard, board-level, hardware interface to the backplane. This interface to the backplane and other local hardware will then depend on the design of a particular board. The CSR standardization effort, which is affiliated with the backplane effort, is standardizing some of the board-level register assignments and the NGCR program may standardize others, but there will most likely be many differences between different implementations of a processor board.

10.3.8 *LSCI*

This "interface"

LPOS <-> SRAX

is primarily a set of protocols, data formats, and conventions that provide communication between an LPOS and the SRAX and IRAXes. The functionality provided by this interface, to a large degree, determines the amount of coordination of LPOSes available to the SRAX and IRAX components. This interface may be hidden (i.e., proprietary to the operating system vendor) in a system where the LPOS, IRAX, and SRAX are developed as a single system. The description of this interface may also include the protocols for communication among the IRAXes and the SRAX.

10.3.9 *OSOSI*

This interface

LPOS <-> LPOS

allows one LPOS to communicate with other LPOS instances in the system and allows instances to share resources and to cooperate with each other. This interface is needed if the goals of reliability and dynamic reconfiguration in a heterogeneous system are to be provided by the operating system rather than being available only if supplied by the application software. The scope of the services provided by this interface depends on the level of coordination needed among the separate processor nodes. Note that these interfaces are not application program interfaces although the OSOSI may be necessary to support the functionality provided by some parts of the application program interfaces.

An example of the type of communication needed is when two LPOS nodes share a memory board across the backplane. The two nodes must coordinate with each other to allocate and use the memory. Without an OSOSI, the application developer will have to perform all the memory management coordination in the application. Another example of the kinds of messages needed are those to coordinate the live insertion or removal of processor boards in a running system as supported by the NGCR backplane. This interface is conceptually at a lower level than the LSCI. An LSCI would probably be built, however, using the OSOSI.

10.3.10 *PSEI*

This interface

PSE <-> LPOS

provides a means for the PSE to interact with the LPOS for loading software, testing, debugging, etc. In many systems, this interface would be removed before the system became operational. The standardization of this interface will make it easier to have a common PSE for different LPOS instances. Some systems will have a different, hardware-based, non-intrusive testing interface to the LPOS. The non-intrusive testing interface is, by its design, invisible to the LPOS hardware and software when in operation. The NGCR OSIF Standard may, however, need to address the capabilities required to set up and control non-intrusive testing interface hardware.

10.3.11 *SAPI*

This interface

Source application software <-> LPOS

is the one that is normally thought of when an OSIF is being discussed. This is the interface (or set of interfaces) that the applications programmer uses to develop embedded systems. This is the high-order language bindings (Ada, etc.) to the OS system calls. A BAPI is a compiled, binary version of this interface.

10.3.12 *MMI*

This interface

User <-> application

is between the application user and the application programs. The hardware used for this interface is often some combination of special-purpose display devices and user-input devices. This interface can be considered a special form of device interface for which a standard set of device-driver commands would be useful to promote software transportability and easy movement of development engineers among projects and users among embedded systems. Any standardization at this level may come out of the NGCR GLIWG. Many existing operating systems have very little support for the MMI, except for (perhaps) a command-line parser.

10.4 OPERATING SYSTEM SERVICES

This section describes the major groups of operating system services that may be required of the NGCR OS. Not all of these services require a programming interface; therefore, the services can be described as either explicit or implicit services. Explicit services are those that can be accessed from an application program (by means of the API) and, generally, are only provided when requested. Implicit services, on the other hand, are services that the OS provides without a direct request. An example of an implicit service is the prevention of one program from writing over the memory of another. An example of an explicit service is a call to an OS routine to output a block of memory to some device.

The OS services often are available at or support more than one of the interfaces described in section 10.3. Each of the services in this section lists the associated interfaces.

10.4.1 *Architecture-Dependent Services*

The service's interfaces are LDI, LHWI, LANI, GRKI, and OSOSI.

These services allow the system to interface with non-NGCR resources such as computers, networks, and operating systems. This will facilitate portability, technology insertion, and the reality that the NGCR system will need to interface with many existing systems that will be in use for many decades to come. These services may allow the system to interface with today's Navy standard computers such as the AN/UYK-44 or AN/UYK-43 as well as special-purpose computers such as the AN/UYS-1.

This set of services also includes a means to access the special hardware features of a system and is, in effect, a standard way to access nonstandard functions

10.4.2 *Capability and Security Services*

The service's interfaces include all interfaces.

These services support the ability of the system to control usage so that system integrity is protected from inadvertent or malicious misuse. These protection services provide a mechanism for the enforcement of the policies governing resource usage. Note that many of the security services are implicit services, i.e., they are provided without an explicit request to the operating system. There are two distinct classes of system access with which operating system services must be concerned: physical access and logical access.

Security services at the physical level are used to protect against security compromise, given that unauthorized personnel may have physical access to system hardware. Typically, the physical access is to a terminal or terminal/display cables or both; however, physical access may also include network cables, central processing units, disk drives, or tape drives. Different types of physical access by unauthorized personnel may require different operating system services and/or hardware to support secure operation. For example, if unauthorized personnel have physical access to the network cable, then services may be needed to support the encryption of any secure data passing through the network. This is because a person may hook a data reading device to the cable without needing to get access by use of the operating system.

Logical access is the ability to interact with the operating system by use of a terminal/display. Security services at the logical level can be implemented through passwords and watchdog timers.

Capability services attach operation lists that limit the ability of the processes to act on resource objects. This is to ensure that the resources are not misused. Access to resources can be protected by services using capability lists as well as access lists, lock/key mechanisms, global tables, or through dynamic protection structure services.

10.4.2.1 *Prevention of Unauthorized Access.* Access to the system may need to be guarded from attempted access by unauthorized personnel. The points of

access to the operating system of concerned are through the SAPI or PSEI. Given the mode of operation (system high, multilevel, open) at which the system is operating, these services differ and have differing implications on other system services (such as reliability, naming, etc.) and system performance.

10.4.2.2 *Prevention of Data Compromise.* These services prevent access of data by users not authorized to the data. These services may be implemented using access lists on files (and directories) and/or encryption of data or in other ways.

10.4.2.3 *Prevention of Service Denial.* These services ensure that a service request will be met by the operating system in a reasonable time if the requestor is authorized to use the service. These services ensure that a bandit user or process cannot cause system malfunction by monopolizing system services or resources.

10.4.2.4 *Security Administration.* This category involves services to allow the managing of the security system including the administration of permissions to personnel, data, and services as well as capability lists. In addition, it permits the administration access mechanisms (passwords or whatever) and services that allow the system to switch modes of operation. The services will likely be accessed by the target system operator with security responsibilities through the target system operator services.

10.4.3 *Data Interchange Services*

The service's interfaces are OSOSI and SAPI.

This set of services provides data conversion among different data representations. One scheme for providing these services is to have a single canonical representation for the important data types (integer, real-time, etc.) and then each implementation of the OS or compiler would provide conversion functions between the canonical representations and its own internal representations.

10.4.4 *Event and Error Management Services*

The service's interfaces are ARTEI, DBKI, GRKI, LANI, LDI, LHWI, MMI, OSOSI, PSEI, and SAPI.

These services provide a common facility for the generation and communication of asynchronous events among the system and application programs. A major use of the event services is to report error conditions, but they may be used by device drivers and the OS to provide an indication of some condition to the application programs.

10.4.5 *File Services*

The service's interfaces are BAPI, DBKI, LDI, LANI, OSOSI, and SAPI.

These services allow the system and applications to create permanent storage locations for data, which are stored on files; the files are organized in directories. Files are managed and accessed through logical names by the many system components that use the files, such as the application, target system operator, and program support environment.

10.4.5.1 *Naming and Directory Services.* These services allow the access of files and directories through logical names rather than the actual hardware device naming conventions. The services may allow sharing of files at various levels. For example, the services may not allow any shared naming of files and directories between systems; they may allow shared files by explicit naming; or they may allow shared files by implicit naming. The directory services present a view or views of the directory structure to the application or target system operator.

10.4.5.2 *Real-Time Files.* Real-time systems often need special files to ensure fast, predictable, and consistent performance in time-critical situations and robustness in the event of system or power failure. The need for a known response time for a given I/O function drives the design of these files and services. One service may preallocate the complete disk space needed for a file at creation time, while another guarantees that records within files are aligned in an optimal way (such as along word boundaries). Services may support the access of records within the file in ways that make response time constant or bounded, such as direct access.

10.4.5.3 *File Modification Primitives.* Primitive services for files and directories include the ability to read a portion of the file, write to a portion of the file, open access to a file, create a new file, close access to a file, and delete a file. These services may be very complex. For example, the access to read or write may be direct (by record number), sequential (one record at a time), or indexed (by a key). In addition, services may be needed to support the merging, appending, splitting, and copying of files. The services may need to support a variety of file structures.

10.4.5.4 *File Support Services.* Additional services support the physical devices on which the files and directory reside. These services include the dismounting/mounting, formatting, and partitioning of media.

10.4.6 *Generalized I/O Services*

The service's interfaces are DBKI, LANI, LDI, LHWI, MMI, OSOSI, and SAPI.

Generalized I/O services provide higher level constructs and functions for doing I/O to devices that do not fit well into the common file I/O paradigm. Nonblocking I/O to devices and files is required in real-time systems. These services include nonblocking I/O and I/O to special devices. In nonblocking I/O, output or input is initiated under program control, but the program continues execution while the transfer takes place. Many special hardware devices may need I/O supervised by the operating system.

10.4.7 *Networks and Communications*

The service's interfaces are BAPI, LANI, LHWI, OSOSI, and SAPI.

These services involve the information exchange between processes within an NGCR system. The processes may be located on different local processor nodes or on the same node.

10.4.7.1 *Network Control and Status.* These services provide authorized users with the capabilities to determine the status of communications facilities (including networks) and to control communication subsystem working parameters. Many of these services logically reside in the LPOS. Other control and status services, however, logically reside in the SRAX. These services include system startup configuration, network restart, network initialization, network security, network scheduling, network monitor, network configuration management, and time management. They provide services that allow the communication subsystems to efficiently use their resources. These services may make use of the OS scheduling services described in the scheduling services section, section 10.4.12.2.

10.4.7.2 *IPC.* This service allows a local processor node's local operating system to request a procedure, a function, or a transaction to be performed on some processor node or logical resource. There are various forms of IPC, some of which specify the receiver, some specify the sender, some are synchronous (i.e., delay the sender until the communication is completed), some are asynchronous, etc. The particular forms that need to be specified by the NGCR OSIF Standard are to be determined.

10.4.7.3 *Distributed Voting.* This service allows the application to request and collect "votes" or answers from applications distributed across some communication medium. The request may require processing and/or information from the voters, and the answers returned by the voters may be simple (yes/no) or complex. The resulting votes will be analyzed by either distributed voting services, other services (such as reliability services), or application program(s). Distributed voting services will handle situations where a vote is not received in the appropriate time. This service will likely make use of the synchronization and time services.

10.4.7.4 *Remote Resource Allocation.* This service allows a local node to allocate for usage a resource that is physically located at another node within the set of cooperating LPOSs or within the set of processors controlled by an SRAX-level operating system.

10.4.7.5 *Naming.* These services allow the usage of system resources through logical names rather than through the actual hardware device-naming conventions. Furthermore, they allow the resources of other processor nodes to be accessed by use of a a logical name so that no knowledge of the resource's location is needed (the resource's location may change over time). Logical names are also used by security services to hide resources from unauthorized processes by only letting authorized processes know the logical name that is needed to use the physical resource. The logical name to physical name relationship can be one to many, many to one, or many to many. Many times, one physical resource may have multiple logical names as well as one logical name representing a "bank" of available physical resources. These services must provide the proper resolution of names, logical and physical, in all of these cases.

10.4.8 *Process Management Services*

The service's interfaces are ARTEI, DBKI, OSOSI, PSEI, and SAPI.

Typically, the following process management services are required by application programs:

- Create a process and make it ready for execution
- Destroy a process and recover its resources
- Evaluate a reference to a process
- Evaluate a connection to a process, where a connection is a logical communication path between any two processes.

10.4.9 *PSE Services*

The service's interface is PSEI.

During the system development process, there is a need for the PSE to communicate with the system under development. The operating system in the target will need to support that communication. These services may not be available at the API but may be accessed by means of a different interface. These services may also be removed from the system when it is deployed. The types of services included here are down-loading of compiled programs and data into the target system, uploading to the PSE of program results and trace information, and the interactive debugging by a developer on the PSE of an application running on the target system

10.4.10 *Reliability, Adaptability, and Maintainability Services*

The service's interfaces are BAPI, DBKI, LDI, LANI, LHWI, OSOSI, and SAPI.

Many times, robustness, or fault tolerance, of a system or application is a desirable feature. The services supporting robustness (reliability, adaptability, and maintainability) are often implied services in that there is not a direct interface to these services through the SAPI layer of the operating system. Reliability and adaptability services deal with the need for the system to perform functions that the application requests in a timely manner, whenever possible. Reliability is the ability to correctly perform a job to completion; adaptability is the ability to change the system's logical makeup (or jobs to do) over time; and maintainability is the ability to keep the system in operating condition. A highly adaptable system can facilitate the reliability of application's functions.

10.4.10.1 *Fault Tolerance Services.* These services allow the system to react to the loss or incorrect operation of system components at various levels (hardware, logical, services, etc.). The classical model of fault tolerance has a three-step approach. The three steps are fault detection, fault isolation, and fault recovery. Typically, implementations divide these steps into substeps or integrate them into one or two steps. Additionally, fault diagnosis services support the other steps in the treatment of a fault.

Various fault tolerance strategies, such as checkpointing and voting, are implemented as a collection of services comprising one or more of the steps in the classical fault tolerance model. For example, services involved in implementing a three-node voting scheme will include a vote comparator service (fault detection), vote analyzer service (fault isolation/fault diagnosis), a service to pass the majority "answer" through (fault recovery), as well as a service to disable the faulty resource and reconfigure the voters (fault recovery/reconfiguration).

Service categories "fault tolerance" and "event and error management" may share services with each other.

10.4.10.1.1 *Fault Detection.* Fault detection services are concerned with determining when a fault has occurred in the system. Fault detection services are both passive and active. Active services are those that attempt to determine the status of various system components by testing those components. Passive services, on the other hand, try to ascertain system components by passively gathering information and watching the behavior of the system.

10.4.10.1.2 *Fault Isolation.* Fault isolation services attempt to determine the component at fault and segregate the faulty component from the rest of the system. Services may be shared between the fault detection and isolation service library in that they perform both functions.

10.4.10.1.3 *Fault Recovery.* These services attempt to bring the system into a consistent state. These services may be very interrelated to the scheduling services, network services, and data base services depending on the recovery scheme used. Redundancy of resources is needed many times to support fault recovery. Resources may include data, process, processor, disk drive, etc. As parts of the system fail, it may no longer be possible to satisfy all the requirements of the application. Services to support graceful degradation may be used to ensure that critical activities do not fail.

10.4.10.1.4 *Fault Diagnosis.* These services deal with the system's ability to analyze the attributes of a system fault and determine its cause. These services tend to be very interrelated with fault detection and fault isolation services.

10.4.10.2 *Fault Avoidance.* These services involve the avoidance of faults before a failure in the system component occurs. If a system can detect that the operation of a component is approaching the edge of its operational range, then a standby or backup component could be phased in to replace it. Another form of fault avoidance is logging of shocks, temperature extremes, etc., so that it can be predicted that a component will not meet its original, expected service life.

10.4.10.3 *Software Safety.* These services involve the system's ability to keep application software from causing harm to the system's software, hardware, or user. For instance, a process may attempt to write into another process's memory space without permission. A good example of a reliability method that may provide software safety is a bounds checker. The checker compares an answer supplied against the bounds. If it is not within the bounds, the bounds checker will not allow the answer to propagate, possibly causing damage to the system's integrity. Additionally, it may send a fault message (or security violation information, depending on the type of answers expected) to the proper service. To enhance software safety, other services and processes should be only given the resources necessary to complete their job.

10.4.10.4 *Status of System Components.* These services involve the obtrusive and non-obtrusive diagnosis of the state of system components. (For further explanation of these services, see section 10.4.10.) Additionally, these services may need to record and/or display information concerning performance, configuration, and general system information.

10.4.10.5 *Reconfiguration.* These services allow the system to reconfigure its view of the world. These services allow the system to substitute different resources to perform system functions such as substituting a new physical I/O channel to support a logical channel. These services are part of the API, but their use may be restricted to specially authorized programs such as those used by the target system operator.

10.4.10.6 *Maintainability.* These services provide support for the maintenance of the embedded system. A major component of that support is the collection and logging of information about the operation of the system. Typical information to be logged is as follows:

- Software and hardware errors during operation
- Processes that failed or almost failed to meet scheduled deadlines
- Performance metrics for system tuning
- Times when the system operated in extreme environmental conditions
- Errors reported during startup self-testing
- Attempts to violate rules of the systems security policy.

10.4.11 *Resource Management Services*

The service's interfaces are ARTEI, DBKI, LANI, LDI, LHWI, OSOSI, PSEI, and SAPI.

These services are involved in the management of the systems resources, which include CPU, memory, I/O, and other physical devices. Services that manage the usage of the CPU are described in section 10.4.8. Services that manage the usage and access to files are described in sections 10.4.5 and 10.4.6. The scheduling of the system's resources is described in section 10.4.12.

10.4.11.1 *Memory Management Services.* These services support the usage of the LPOS main memory(s). These services supply a virtual view of the memory or memories on the computer as seen by applications and perform the proper mapping of virtual to physical memory (performing any swapping of memory paging needed in the process). Memory management services provide storage to allow process and data migration as well as initialization. The memory manager many times receives requests for service from the process management services to allocate and deallocate memory for process usage. The major services of memory management fall into five categories: allocating physical memory, mapping of logical address to physical storage, memory sharing, extending memory (virtual storage), and protecting user information.

10.4.11.2 *Device Management Services.* These services attempt to remove the dependencies on physical resources. The service user sends information to and from the devices by way of logical data structures or device service requests or both. These services mainly serve to supply four functions: device allocation, device control, device status, and device access.

10.4.12 *Synchronization and Scheduling Services*

The service's interfaces are ARTEI, LANI, LDI, LHWI, OSOSI, PSEI, SAPI, and DBKI.

10.4.12.1 *Synchronization Services.* These services are involved in the ability to synchronize the operations of other services, functions, processes, and/or resources. Services such as distributed voting and remote resource allocation will need to use these services to accomplish their required functionality. Synchronization services are needed for both the LPOS's operation and the control of the distributed system. Synchronization services may need to use system monitoring services to adjust to system changes.

10.4.12.2 *Scheduling Services.* These services schedule or arbitrate the usage of various resources of the NGCR OS, particularly the CPU. The scheduling services must be able to queue up requests to use a particular resource. This situation is made more complicated by the common need to schedule processes to run cyclically at a fixed period. When a resource become idle, the scheduler must select one of the "requestors" of the resource to grant use of the resource. These services are listed separately rather than under the services that use scheduling to emphasize that there should be uniformity and consistency of scheduling across the range of resources.

Typically there are at least two types of scheduling occurring in an operating system: short term and long term. Long-term schedulers determine which possible requestors at a given time may actually request a resource. The short-term scheduler selects from among the active requestors who currently have need of the resource and allocates the resource to the selected requestor. For example, if the requestors are processes and the resource is the CPU, then the long-term scheduler manages the movement of processes from inactive (waiting in batch queues or in hibernation) to active (in wait or execute). The short-term scheduler, on the other hand, would determine which process should execute next on the CPU. Hybrid services between the two may also be available in the operating system.

When a request for a resource is submitted to the operating system (at some local operating system node), it is not always serviced at that local node. The most advantageous way to service the request may result in part or in all of the work being performed at a different processor node. Several reasons may cause this to occur including load balancing, resource availability, computation speedup, hardware preference, and software preference. These services may hide from the application the fact that the functionality was being performed at a different node. This has the advantage that the code needs to know little about the system on which it is running. Alternately, the services may allow the user to specify directly on which logical resource the function should be executed.

The priority scheduling of resources allows the requestor to have associated with it its importance to use the service. More complex schemes

also have a criticalness of the request that is used for graceful degradation purposes. The scheduler(s) will use the priority information to arbitrate resource requests and to queue requests in the specific order. A priority scheduler may need to support multilevel queues to support proper execution.

Preemptive schedulers will deallocate a resource from a requestor when certain events occur. Usually this is when a requestor of a higher priority or importance requests the resource or when a specified time limit for the resource has expired.

10.4.13 *System Initialization and Reinitialization Services*

The service's interfaces are DBKI, GRKI, LANI, LDI, LHWI, OSOSI, PSEI, and SAPI.

System initialization includes a complete restarting of the software, starting up the attached hardware subsystems devices, doing subsystem and system self-tests, and completely initializing the data base.

System reinitialization includes restarting the software while using the existing data base information. The software may have to be reloaded and the database may have been reestablished by a system recovery. Attached hardware subsystems may also need to be reinitialized.

Reinitialization should include a function to restart applications redistributed to other processors after a processor module failure. Within a processor, there should be a function to initialize applications in a system with the existing software but with the data base reinitialized. Also within a processor, there should be a function to restart the applications in a system with the existing software and data base retained.

10.4.14 *Time Services*

The service's interfaces are ARTEI, LHWI, OSOSI, and SAPI.

The following time management services are likely to be needed:

1. Local time of day, which includes the time based on a 24-hour or 12-hour clock.

2. Measurement of elapsed time.

3. Distributed time that would be a capability to coordinate local time of day maintained by any LPOS.

4. Requests that a process be delayed for a specified elapsed time.

5. Requests that a process be delayed until a specific time.

6. Requests for process notification at a specific time or after a specified delay.

10.4.15 *Ada Language Support Services*

The service's interface is ARTEI.

These services support the use of the Ada programming language. This section describes some special needs of the language that may be addressed by services within the system.

10.4.15.1 *Full Ada Language Support.* While an NGCR OSIF Standard compliant operating system may be implemented in various languages, it should support the execution of programs written in Ada. At the least, this means that the operating system together with the compiler's runtime library should include all necessary parts of an ARTE.

For highest efficiency, some parts of the ARTE should be an integral part of the operating system although the interface definition itself need not depend on that integration. An example of an integration problem is that of task scheduling. Many current implementations of an ARTE "on top of" an existing operating system schedule Ada programs as single entities. If an Ada task is running and becomes blocked, the OS does not consider other tasks in that program for execution, but only considers other programs.

10.4.15.2 *Exception Propagation to OS.* When an exception from an application or hardware event is propagated to an operating system, this service handles the communication of the event to the proper application routine.

10.4.15.3 *Interrupt to Process Mapping.* When an interrupt occurs, this service will ensure the correct mapping from interrupt to Ada task is made (even when the interrupt and tasks are located on separate processor nodes of an SRAX level distributed operating system).

10.4.15.4 *Priority.* These services support the full priority semantics of the Ada tasks.

10.4.15.5 *Rendezvous.* These services support the rendezvous of tasks (from tasks being implemented as within one single process to tasks being implemented as distributed processes).

10.4.16 *Data Base Services*

The data base management system in an embedded system has several functions, including access control, consistency checks, maintaining consistent copies for fault tolerance, and security. The need for data base services as part of the OS arises because of the interaction of the DBMS need for performance and multilevel security needs. If parts of the OS are part of a TCB for a multilevel secure system, then the lower level parts of the DBMS

(the data base kernel) will have to be part of that TCB or be built "on top of" the OS. The data base services may be specialized services for use to support a DBMS or they may be of general use for application programs. The NGCR Data Base Working Group has the responsibility for defining interfaces for a full DBMS. Those applications that do not need a full DBMS may have need for simple data base services in connection with the file system services. These services may not be part of the NGCR OSIF Standard but may be provided on a case-by-case basis by the implementors of integrated DBMS/OS systems.

10.4.17 *Graphics Kernel Services*

The service's interface is GRKI.

This interface provides low-level access to graphics services. The NGCR GLI may require low-level access for performance reasons. It is not clear at this point whether the API will be sufficient for the needs of the GLI implementation.

10.4.18 *LPOS-to-LPOS Communication Services*

The service's interfaces are OSOSI and SAPI.

These services support a standardized way of passing information between LPOSes of NGCR operating system(s). It is to be determined whether LPOSes within an NGCR operating system implementation can be supplied by different vendors and plugged into the NGCR operating system. If this is the case, services must support a common set of protocols so that the LPOSes of the system are able to coordinate services and resources with each other. This coordination may be done all at the application level with various application programs communicating with each other through their individual LPOS. This would be fully supported by the SAPI. Alternately, the LPOSes could also communicate and coordinate resource usage directly with each other by means of the IRAX-SRAX hierarchy. This would provide the primitives for such services as automatic dynamic load leveling and automatic reconfiguration. In either case, the support of LPOS-to-LPOS communication will require a common set of OS-level messages and protocols whether it is provided by the application, by agreement of the vendors, or as part of the NGCR OSIF Standard.

10.4.19 *MMI Services*

The service's interfaces are MMI, SAPI, and PS.

These services allow I/O to be interchanged between the system and the user of the embedded system through the SAPI or PSEI in an efficient and standardized way. Services that may be included are menu services, windowing services, command-line services, parsing services, and pointer device services. These services will interface with low-level device services as

needed, while presenting a higher level view to the human user. Higher level interfaces for much of this set of services may be provided by the GLIWG of NGCR rather than by the OSSWG.

10.4.20 *Target System Operator Services*

The service's interfaces are SAPI and PSE.

The system operator needs to access and control the operating system to allow the system to perform properly. If a system has an operator, the major functions that need to be supported are system control, reconfiguration, and status reporting. These services today are usually made available to an operator through a command language interpreter, which is an application program that calls these OS services.

Note that the MMI services provide the building blocks (menu utilities, command parsers, etc.) for building the user interface while the target system operator services make available system status and control functions to appropriate application programs with the proper security level.

10.5 TARGET DOMAINS

The domains in which the NGCR OSIF Standard are expected to be used vary greatly and in several different ways. This section discusses important ways that the target systems differ and the major implications that these differences may have on an operating system and on the NGCR OSIF Standard. It is expected that there will be a need to tailor or subset the OSIF Standard for particular target systems. One concept is that the OSIF Standard may not be one standard but a family of standards, each member engineered for a particular class of target systems but having much in common with other members of the family of standards. Another concept for tailoring is that there will be only one complete set of interfaces implemented with a small OS core or kernel and a compilation system that links in with each application only the functions or services needed by that application. A third tailoring method could be to expect the OS vendors to provide an OS tailoring tool so that a general OS can be tailored by the system engineer to fit the needs of a particular target system.

Note that there are multiple dimensions along which OS needs vary. This implies that there is no simple way to create a family of OS standards that meets the needs of the different target domains and is also small in the number of member standards.

10.5.1 *Target Processor Interconnection*

The processor interconnection, i.e., the means of communication between processors, can vary from a single processor system with no connection to other processors to an LPOS system connected by means of a full network to many other processors of various types.

10.5.1.1 *Single-Processor Systems.* In a single-processor system, the OS has little if any need to support network communication services except as an alternate API for IPC. If there is more than one program on the processor, then some form of interprocess communication will often be needed to allow the application programs to coordinate with each other.

10.5.1.2 *Multiprocessor Systems.* A multiprocessor system is one where multiple processors share common memory. The sharing of common memory allows the implementation of fast synchronization as well as fast communication between processors. Many multiprocessor systems have local (or private) memory in addition to the shared memory. This can increase the efficiency of execution of programs, but having to support two varieties of memory can add to the complexity of the OS.

10.5.1.3 *Distributed Systems.* A distributed system is one that has multiple processors without any shared memory. There are three kinds of distribution, but real systems are often complex combinations of the three kinds of distributed and multiprocessor systems. The kinds of distribution are classified according to the interconnection mechanism as follows:

- Backplane interconnection
- Local area network interconnection
- Full network interconnection.

10.5.1.3.1 *Backplane Interconnection.* Processors that are interconnected by a high-speed backplane have a fast communication mechanism, but depend on the capabilities of the backplane hardware for synchronization and communication. The operating systems in a backplane interconnected system need to use more complex synchronization techniques that tend to be somewhat slower than in multiprocessor systems; however, the SAPI for those services may not differ.

10.5.1.3.2 *LAN Interconnection.* In an LAN interconnected system, the processor-to-processor communication is by means of the LAN. This kind of distribution is limited to processors that are on the same local network without any storage of messages between nodes of the network. In this type of system, the reliable communication needed for IPC can be achieved by use of simple send and acknowledge schemes, although the communication speed adds significant overhead time compared to backplane interconnection.

10.5.1.3.3 *Full Network Interconnection.* In a full network interconnected system, there are multiple LANs or at least multiple LAN segments with bridges and/or gateways that use store and forward communication schemes. The lack of a direct network connection between processors significantly increases the complexity and overhead required to produce a reliable IPC. Depending on the organization of the SAPI, the difference in complexity may not be visible to the application designer except in longer communication delays of a particular implementation.

10.5.2 *Security*

A major goal of the NGCR program is to provide systems that can be used to meet the security needs of the Navy. The security needs of a project vary with the project and can be met in a variety of ways. For this model, the security needs will be grouped according to the type of access control required.

10.5.2.1 *Targets with No Security Requirements.* Many Navy systems have no special security requirements either because they do not process classified or sensitive material or because they operate in facilities that meet the security requirements with physical security alone. For these systems, the OS would probably not implement the security services. For compatibility with secure systems, however, there may be limits on the functionality provided by the OS through the API.

10.5.2.2 *Targets with DAC Requirements.* Many non-embedded computer systems provide some form of DAC, usually by means of passwords and file permissions. DoD 5200-28.STD defines the classes of computer security or trustedness provided by computer systems. Class C systems provide discretionary access control, which is a means of restricting access to objects based on their identity or on the identity of the groups to which they belong. Supporting discretionary access controls requires that there be interfaces to the OS by which the permissions, passwords, etc. can be changed. Also required is a means of reporting or processing access violations.

10.5.2.3 *Targets with MAC Requirements.* MAC is a means of restricting access to objects based on the sensitivity of the information contained in the objects and in the formal authorization of subjects to access information of such sensitivity. MAC adds only minor interface requirements but can significantly increase the amount of implicit services required of the OS.

10.5.3 *Robustness*

Robustness refers to how well the system can be expected to continue operation and how quickly it can be repaired when some part malfunctions. It also refers to features of a system that prevent unsafe actions from taking place.

10.5.3.1 *Reliability and Availability.* Reliability and availability can be achieved by fault avoidance or fault tolerance or both. Fault avoidance is achieved by increasing the reliability of the hardware components and applying conservative design practices. Fault avoidance is primarily achieved by means of hardware design and formal methods for software design. Fault tolerance is achieved by the use of redundancy and requires software support.

Fault tolerance is concerned with both data integrity and processing integrity. Data integrity deals with providing survival of data when components fail and is closely linked with data base system technology (data replication, atomic transactions, etc.). Processing integrity tries to ensure correct and continuous processing across instances of component failure. Processing integrity is especially important in safety-critical real-time systems.

10.5.3.2 *Software Safety.* Software safety has the goal of preventing any unsafe action even in the face of incorrect software processing. Software safety often involves independent monitors (either software or hardware) that check the "reasonableness" of results or actions of the system.

10.5.3.3 *Maintainability.* Maintainability of a system describes how quickly and correctly system errors or failures can be determined and corrected. The OS can provide services that aid in maintainability including the logging of system errors (hardware and software), reporting of built-in test results and usage logs for scheduling preventative maintenance.

10.5.4 *Richness of the Set of OS Services*

The target domains of the NGCR OSIF Standard vary in the number of OS functions needed. A small, single-purpose system or one with an extreme emphasis on performance may need a very small, finely tuned OS; a general-purpose system may need many services to be provided by the OS. This is somewhat a matter of philosophy. One view is that the NGCR OSIF Standard should be a minimal set of interfaces, leaving to the particular project the job of implementing higher level functionality in a project-specific manner. Another view holds that the NGCR OSIF Standard should be a full set of interfaces so that there can be a large amount of software portability among projects.

10.5.5 *Real-Time Requirements*

Real-time computing has time constraints, i.e., computing that involves intricately intertwined computation deadlines (often imposed by external stimuli) on short time scales. While the NGCR OSIF Standard are for embedded systems, not all embedded systems have real-time requirements, and those target systems with real-time requirements have various needs.

10.5.5.1 *Non-Real-Time Target Systems.* Some embedded systems are used for applications that have no stringent time demands. Examples of these systems include those used for planning and for maintenance support. These systems still need an efficient, high-performance operating system but the correctness of computational results do not depend on their being available at a particular time.

10.5.5.2 *Real-Time Target Systems.* Real-time systems are probably the most common targets for the NGCR OSIF Standard compliant operating systems. Therefore, the OSIF Standard must surely support the needs of embedded real-time systems. Common OS real-time requirements include the ability to specify a deadline for completion of a process, the ability to specify that a process is to be run cyclically with a specific period, the ability to specify that one process or program is more important to the system than another, and the ability to rearrange the importance of processes or programs as the operating mode of the system changes.

10.5.5.3 *Critical-Time Target Systems.* Critical-time targets are those that have real-time response requirements that, if not met, result in system failure. These systems may even require that results not arrive too early, but precisely when they are needed. Examples of these target systems include many safety critical systems such as flight control systems. To support these target systems, the OSIF Standard will need to allow flexible and predictable scheduling.

APPENDIX

OSIF GENERAL REQUIREMENTS AND INTERFACES

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

20.1 GENERAL REQUIREMENTS

20.1.1 *Scope*

20.1.1.1 *Definition.* The OSIF shall provide interfaces sufficient to support a wide range of Navy target applications.

20.1.1.2 *Metric.* To be determined.

20.1.1.3 *Rationale.* It is intended that the OSIF will be used by applications from missile guidance systems to large command and control systems to completely integrated platforms. This range of target applications is very demanding, and it is known that there is no one operating system that can satisfy all possible application systems. The goal of the NGCR OSIF is to satisfy the needs of this range of application domains. Therefore, the concerns of this range will take highest priority in determining the appropriate features of the OSIF.

20.1.2 *Design Objectives*

20.1.2.1 *Definition.* The OSIF should provide interfaces sufficient to promote compatibility, interoperability, transportability, and reusability between applications and maintainability of applications.

20.1.2.2 *Metric.* To be determined.

20.1.2.3 *Rationale.* This requirement addresses the reasoning behind the development of the OSIF. These are the qualities that are desired in applications and that can be promoted by the OSIF. Interoperability is the ability of two applications to share data. Transportability is the ability to move an application from one implementation of the OSIF to another with minimal changes to the source code. Reusability is the ability to reuse portions of one application's source code or other pertinent aspects (e.g., design, tests) in the generation of another application. Compatibility is the general ability of two applications to coordinate with one another in their operation, even if they were not originally designed to do so. Maintainability addresses the qualities that improve the ability to maintain the application.

20.1.3 *Basic Services*

20.1.3.1 Definition. The OSIF should provide simple-to-use mechanisms for achieving common, simple actions. Facilities that support less frequently used features should be given secondary consideration.

20.1.3.2 *Metric.* To be determined.

20.1.3.3 *Rationale.* The OSIF should be understandable and usable. Thus, this requirement encourages the selection of a set of interfaces where the frequently used ones (at compilation time) are simple to use, possibly at the expense of less frequently used facilities being more difficult to invoke. This requirement also suggests concentration on supporting those application actions that are likely to have the broadest use.

20.1.4 *Architecture Independence*

20.1.4.1 *Definition.* The OSIF shall be machine-independent and implementation-independent. The OSIF shall be implementable on a wide variety of processors, configurations, and architectures.

20.1.4.2 *Metric.* To be determined.

20.1.4.3 *Rationale.* The OSIF must depend on no properties of specific computers and on no properties of specific implementations. The features should also be chosen to have a simple and efficient implementation in many machines and hardware architectures and configurations (including distributed configurations). Transportability can only be achieved where the OSIF itself can be implemented on a wide range of machines without revealing or relying on machine or implementation dependencies.

20.1.5 *Modularity*

20.1.5.1 *Definition.* The OSIF should be partitioned so that the partitions can be understood independently.

20.1.5.2 *Metric.* To be determined.

20.1.5.3 *Rationale.* This criterion promotes understandability and permits application writers to employ a subset of the OSIF, which will be important for many applications. It should be noted that there can be multiple versions of some partitions within the standard. Independent understanding implies that there should be no undocumented dependencies between partitions.

20.1.6 *Extensibility*

20.1.6.1 *Definition.* The OSIF should facilitate development and use of extensions of the OSIF; e.g., OSIF interfaces should be composable so that they can be combined to create new interfaces and facilities, or it should be possible to add new interfaces for new functions.

20.1.6.2 *Metric.* To be determined.

20.1.6.3 *Rationale.* The state-of-the-art and the state-of-the-practice in computer technology are rapidly changing. It is impossible to fully list all interfaces that will be required in all future application domains. Therefore, the list of specialized interfaces must be extensible.

20.1.7 *Uniformity*

20.1.7.1 *Definition.* The OSIF should be based on a consistent set of unifying well-defined conceptual models. All OSIF features should uniformly address aspects such as status return, exceptional conditions, parameter types, and options.

20.1.7.2 *Metric.* To be determined.

20.1.7.3 *Rationale.* The design of the OSIF should minimize the number of underlying concepts and unifying principles. A unifying principle is a model that unifies (a subset of) the interfaces. It should have few special cases and should consist of features that are individually simple. These models should also be consistent with one another. However, these objectives are not to be pursued to the extreme of providing inconvenient mechanisms for the expression of common, reasonable actions.

20.1.8 *Completeness*

20.1.8.1 *Definition.* The OSIF should provide a complete set of facilities for all elements of its underlying conceptual models.

20.1.8.2 *Metric.* To be determined.

20.1.8.3 *Rationale.* Because one of the major goals of the OSIF is transportability, it must provide a sufficient set of facilities to support applications so they do not have to use facilities outside of the OSIF.

Although it is desirable that the OSIF be complete and provide all facilities for applications, a requirement that mandates all facilities for all applications is recognized to be unachievable in practice. The goal for OSIF should be to optimize the degree of completeness, compromising between all possible facilities and those that can be implemented widely. The things within the conceptual models of the OSIF can, in all probability, be manipulated only by the OSIF interfaces. Hence, all desired manipulations must be catered for by the OSIF. Simple manipulation examples are:

1. If there is a facility to create something, then there should also be a facility to delete it.

2. If there is a facility to set a value, then there should also be a facility to examine it.

20.1.9 *Language Independence*

20.1.9.1 *Definition.* The OSIF should include a clear description of its interfaces that is independent of any particular programming language binding.

20.1.9.2 *Metric.* To be determined.

20.1.9.3 *Rationale.* Although Ada is the language of primary interest to the Navy, other languages will also be important . For example, the NGCR objective of being able to purchase commercial off-the-shelf components often involves such popular languages as "C" and Navy systems in the 21st century might well incorporate intelligent subsystems that are written in popular artificial intelligence languages. The best way to evolve a standard that can withstand such demands is to develop the services in a language-independent way, thus allowing the development of any number of compatible language bindings. To achieve this, the basic description must be complete, consistent, unambiguous, and abstracted away from the details of any particular programming languages, but capable of accommodating a variety of languages.

20.1.10 *Ada Language Binding Syntax*

20.1.10.1 *Definition.* The OSIF shall have an Ada language binding consistent with the language independent model. The OSIF Ada binding syntax shall be expressed as Ada package specifications, as defined by the "Ada Language Reference Manual" (ANSI/MIL-STD-1815A-1983). (This will be referred to as the Ada LRM for the remainder of this appendix.) It shall provide a fully documented interface from Ada to all operating system facilities for which there is no appropriate Ada language construct.

20.1.10.2 *Metric.* To be determined.

20.1.10.3 *Rationale.* The interface should be fully specified in Ada (possibly in addition to other languages) to prevent ambiguities from arising concerning how the specification language maps to Ada. All ambiguities in specification will result in reduced portability. There will be many interfaces that provide facilities that are not directly supported by the Ada language, and it will be necessary for an application to have access to these. It must be easy for the Ada programmer to gain access to these facilities.

20.1.11 *Other Language Binding Syntax*

20.1.11.1 *Definition.* The OSIF should have a variety of language bindings, consistent with the language independent model. The syntax of each shall be presented in a manner consistent with good practice for that language.

20.1.11.2 *Metric.* To be determined.

20.1.11.3 *Rationale.* A number of language bindings other than Ada will also be desirable. Each should exhibit good style for that language and be presented according to the accepted standard for the language, if such a standard exists.

20.1.12 *Language Binding Syntax Uniformity*

20.1.12.1 *Definition.* Each OSIF language binding should employ uniform syntactic conventions and should not provide several notations for the same concept.

20.1.12.2 *Metric.* To be determined.

20.1.12.3 *Rationale.* OSIF language binding syntax issues (including, at least, limits on name lengths, abbreviation styles, other naming conventions, and the relative ordering of input and output parameters) should be resolved in a uniform and integrated manner for the whole OSIF language binding. Understandability and usability of the OSIF are the intents of this criterion. Users should not have to unnecessarily learn different syntactic approaches for using different OSIF features. The use of several notations for the same concept leads to confusion, and non-uniformity is a recipe for errors in use and difficulty in production of applications.

20.1.13 *Language Binding Syntax Name Selection*

20.1.13.1 *Definition.* The OSIF language bindings should avoid coining new words (literals or identifiers) and should avoid using words in an unconventional sense. Identifiers (variable names) defined by the OSIF language bindings should be natural-language words or industry-accepted terms whenever possible. The language bindings should define identifiers that are visually distinct and not easily confused. The language bindings should use the same name everywhere in the interface set, and not its possible synonyms, when the same meaning is intended.

20.1.13 2 *Metric.* To be determined.

20.1.13.3 *Rationale.* Understandability of the OSIF specification is the intent of this criterion.

20.1.14 *Syntactic Pragmatics*

20.1.14.1 *Definition.* The OSIF language bindings should impose only those restrictive rules or constraints required to support the design objectives (refer to section 20.1.2).

20.1.14.2 *Metric.* To be determined.

20.1.14.3 *Rationale.* Although it would be ideal if no such restrictions were required, practical considerations dictate that some limits will exist in all implementations. Where necessary to support the design objectives, such restrictions should be clearly articulated by the OSIF specification. If they are not necessary to support the design objectives (e.g., if they are present for the convenience of the OSIF or its implementers), then they are not desirable in the OSIF.

20.1.15 *General Semantics*

20.1.15.1 *Definition.* The OSIF should be completely and unambiguously defined. The specification of semantics should be both precise and understandable. The semantic specification of each OSIF interface shall include a precise statement of assumptions (including execution-time preconditions for calls), effects on global data and services, and interactions with other interfaces.

20.1.15.2 *Metric.* To be determined.

20.1.15.3 *Rationale.* This is a call for adequate, usable documentation. It is critical for the ability of independent vendors to develop implementations that can be used readily, both alone and together. Note that the requirement does not prescribe the degree of formality of the language to be used for specifying the OSIF semantics, admitting options from free-form English (as long as it is complete and precise) to a formal semantics specification approach.

20.1.16 *Semantic Consistency*

20.1.16.1 *Definition.* The description of OSIF semantics should use the same word or phrase everywhere, and not its possible synonyms, when the same meaning is intended.

20.1.16.2 *Metric.* To be determined.

20.1.16.3 *Rationale.* The use of synonyms, while desirable in novels, has no place in technical documents and only leads to confusion.

20.1.17 *Error Conditions*

20.1.17.1 *Definition.* The OSIF language bindings shall employ appropriate mechanisms to report exceptional situations that arise in the execution of OSIF facilities. The OSIF specifications shall include error conditions for all situations that violate the preconditions specified for the OSIF interface. The OSIF specification shall define error conditions that cover all violations of implementation-defined restrictions.

20.1.17.2 *Metric.* To be determined.

20.1.17.3 *Rationale.* This requirement demands that the OSIF provide the means for the implementation to inform the user (i.e., an application) whenever an operation does not complete normally; this allows the application to take appropriate action. The use of the Ada exception mechanism is expected for the Ada language binding. However, use of the exception mechanism does not preclude the use of status return values by an Ada binding to provide supporting diagnostic information. Thus, it would be possible to have a single exception representing a number of different (but related) error conditions, with the identification of the specific error condition by use of some status parameter.

20.1.18 *Semantic Cohesiveness*

20.1.18.1 *Definition.* Each OSIF interface should provide only one function.

20.1.18.2 *Metric.* To be determined.

20.1.18.3 *Rationale.* This criterion is a statement of the basic principle of software cohesion. It should be interpreted as making undesirable an interface design with a small number of entry points, with each entry point delivering a range of dissimilar services, and with particular services being selected dynamically by the value of one or more of the parameters. Such a design would almost certainly not provide simple-to-use interfaces for performing simple, common actions. While there can always be instances argued where one person views as several (sub)functions what someone else views as an atomic function, this should be the exception rather than the rule. The overloading of subprogram names allowing the selection of different subprograms depending on the types of the parameters is acceptable. Such an approach has the advantage of reducing the number of unique subprogram names within the OSIF. When used to extreme, overloading can lead to a reduction in the clarity of the OSIF and should be used with care.

20.1.19 *Semantic Pragmatics*

20.1.19.1 *Definition.* The OSIF specification shall enumerate all aspects of the meanings of OSIF interfaces and facilities that must be defined by OSIF implementers. OSIF implementers will be required to provide the complete specifications for these implementation-defined semantics.

20.1.19.2 *Metric.* To be determined.

20.1.19.3 *Rationale.* This calls for the equivalent of appendix F of the Ada LRM, containing a list of specific sections or aspects of the OSIF specification where implementation dependencies (presumably caused by machine dependencies) are allowed to affect OSIF semantics. This list needs to be accompanied by a statement that no other implementation dependencies are allowed other than those listed, and that each OSIF implementation must include documentation stating the implementation characteristics for each item on the list.

20.1.20 *Reaction to Blocking Services*

20.1.20.1 *Definition.* The OSIF shall define what happens if a process calls on a service and that service cannot be completed in a timely manner.

20.1.20.2 *Metric.* To be determined.

20.1.20.3 *Rationale.* An application process may request an OS service that can be initiated immediately, but will not be completed until some later time. Examples would be delay services, I/O services on a device, file I/O services when system buffers are empty or full, and synchronization services. In these circumstances, the OSIF must define the effect on the requesting process and the overall effect on the scheduling of the requesting CPU. Many applications will require blocking of the requesting process, to allow other processes to use this CPU as defined by the scheduling algorithm; other applications may require that the requesting process continue to execute in parallel with the requested service (possibly being notified when the service is complete) or be notified that the service cannot be provided immediately (the application would repeat the request later).

20.1.21 *Bounded Operating Systems Services Times and Context Switching*

20.1.21.1 *Definition.* The OSIF shall support the prediction of operating system service completion times. The OSIF shall be implementable such that these service times are bounded. The OSIF implementer will be required to document the service times.

20.1.21.2 *Metric.* To be determined.

20.1.21.3 *Rationale.* To determine performance with any degree of precision, it is necessary to have a bound on the time necessary for a requested operating system service to complete and to have access to the necessary information for the prediction of completion times. It is also important that the OSIF allow bounded service times that are implementable and predictable for a given OSIF implementation.

20.1.22 *Configurability*

20.1.22.1 *Definition.* The OSIF shall be implementable so that application projects have the ability to configure the implementation to be optimal for the specific application.

20.1.22.2 *Metric.* To be determined.

20.1.22.3 *Rationale.* In certain situations, the user of an operating system will want to emphasize certain aspects of the operating system that, in other situations, the user (or another user) may wish to de-emphasize. Toward that end, the user must be able to configure the OSIF implementation and, for

example, to select from a variety of scheduling options and synchronization mechanisms. There must be minimal penalty in space or time for services not used, and optimization for certain patterns of usage must be available.

20.1.23 *Transaction Scheduling Information*

20.1.23.1 *Definition.* The OSIF shall provide the ability for a process to specify its response requirements for services.

20.1.23.2 *Metric.* To be determined.

20.1.23.3 *Rationale.* Many forms of application processing for hard real-time systems need to be scheduled in a manner necessary to meet the response requirements of all transactions, as well as the response requirement of all the application tasks co-resident with it. This service is not necessarily provided simply by the ability to dynamically set priorities.

20.1.24 *Access Control*

20.1.24.1 *Definition.* The OSIF shall provide a mechanism to allow only certain subjects (i.e., processes) to make use of particular objects (e.g., files, devices). That is, access to certain objects may be limited to only certain application software entities.

20.1.24.2 *Metric.* To be determined.

20.1.24.3 *Rationale.* A means must be provided to limit the access to certain system objects (e.g., files, devices or ports) to only those software entities with sufficient privileges. A mechanism that achieves this on a limited, predefined basis for only some objects would not completely fulfill this requirement.

20.1.25 *Transparency*

20.1.25.1 *Definition.* The OSIF shall provide for the identification of and the access to processes and data, irrespective of their physical location.

20.1.25.2 *Metric.* To be determined.

20.1.25.3 *Rationale.* For many applications, it will be necessary (or at least desirable) to be able to access processes and data by use of logical names rather than by use of physical location. Note that this requirement does not preclude other access methods that do relate to physical location.

20.1.26 *Resilience*

20.1.26.1 *Definition.* The OSIF shall be implementable so that, when physical resources are lost, OSIF facilities that do not depend on these resources may continue to be used.

20.1.26.2 *Metric.* To be determined.

20.1.26.3 *Rationale.* This requirement is intended to preclude the design of an OSIF such that the operation of an implementation must be dependent on the availability of the complete set of resources.

20.1.27 *Network Partition*

20.1.27.1 *Definition.* The OSIF shall be implementable so that partitions of the set (e.g., network) of physical resources can usefully work in isolation and the partitions may be rejoined after recovery.

20.1.27.2 *Metric.* To be determined.

20.1.27.3 *Rationale.* This requirement is primarily concerned with the support for an OSIF implementation that is based on a distributed architecture. An OSIF implementation that could only operate when all processors and interconnections were available would be at a severe disadvantage. If a network fragments, then each segment that has sufficient resources to continue operation should do so. The OSIF must also facilitate subsequent reintegration of the network allowing the partial fragments to be recombined. The OSIF should not be defined such that the only possible distributed implementation would be one that depended on a single system-wide resource for its operation.

20.1.28 *Reference*

20.1.28.1 *Definition.* The OSIF shall provide a means to refer to distinct physical resources (e.g., computational, storage) that are used to implement specific OSIF facilities.

20.1.28.2 *Metric.* To be determined.

20.1.28.3 *Rationale.* Although physical distribution may be transparent to most applications, there are circumstances under which specific parts of applications (most notably those concerned with system status and fault tolerance) may require or have knowledge about the allocation of application components to equipment in the underlying computer configuration. This requirement states that there must be a means for applications to be built to take advantage of such knowledge. Note that these capabilities will always be optional OSIF features in the sense that applications may choose never to reference physical resources and always to leave their mappings (and even the possibility of distributed implementations) up to OSIF implementers. Application writers should also be aware of possible sacrifices in transportability when using such features.

20.1.29 *Reallocation*

20.1.29.1 *Definition.* The OSIF shall provide a means to control (or influence) the manner in which the physical resources are associated with specific OSIF facilities.

20.1.29.2 *Metric.* To be determined.

20.1.29.3 *Rationale.* Certain applications may require particular operations to dynamically control the allocation of application components to underlying equipment in distributed configurations. This requirement states that there must be a means for applications to be built with such capability. Note that these capabilities will always be optional OSIF features in the sense that applications may choose never to reference physical resources and always to leave their mappings (and even the possibility of distributed implementations) up to OSIF implementers. Application writers should also be aware of possible sacrifices in transportability when using such features.

20.2 ARCHITECTURE-DEPENDENT INTERFACES

20.2.1 *Non-NGCR System Interfaces*

20.2.1.1 *Definition.* The OSIF shall support non-NGCR-based systems by providing a subset of its services to those systems. As a minimum, this subset shall include:

- Download, initialize, start, and stop services
- Ability to share resources, particularly peripheral devices

- Process-to-process message communication
- Ability to pass operational status information.

20.2.1.2 *Metric.* To be determined.

20.2.1.3 *Rationale.* The Navy has a large investment in existing non-NGCR-based systems. These systems will continue to be in use for years to come and will need to interface, to some degree, with NGCR-based systems. Additionally, non-NGCR-based systems may need a method to gracefully transition to NGCR-based systems. The NGCR-systems will be required to accommodate interfacing to and evolution of the non-NGCR-based systems.

20.3 CAPABILITY AND SECURITY INTERFACES

20.3.1 *Audit Data Storage*

20.3.1.1 *Definition.* The OSIF shall support the storage and maintainability of audit data.

20.3.1.2 *Metric.* To be determined.

20.3.1.3 *Rationale.* The Trusted Computer Systems Evaluation (TCSEC) (DoD 5200.28-STD) requires storage and maintenance of audit data for all systems at level C2 and above. This requirement does not specify what events are recorded in the log, which is implementation specific.

20.3.2 *Audit Generation*

20.3.2.1 *Definition.* The OSIF shall support generation of audit records.

20.3.2.2 *Metric.* To be determined.

20.3.2.3 *Rationale.* The TCSEC requires the capability to generate audit records for all systems at level C2 and above. This requirement does not specify whether any privileges are required to use this feature.

20.3.3 *Audit Record Contents*

20.3.3.1 *Definition.* The OSIF shall support generation of audit records that uniquely identify the subject, event, and object being operated on.

20.3.3.2 *Metric.* To be determined.

20.3.3.3 *Rationale.* The TCSEC requires that data in the audit trail contain specific information that is used to identify the subject (e.g., process and user ID), object (e.g., data file), and the event that caused the audit record to be generated.

20.3.4 *Audit Data Manipulation*

20.3.4.1 *Definition.* The OSIF shall support the manipulation of audit data.

20.3.4.2 *Metric.* To be determined.

20.3.4.3 *Rationale.* The TCSEC requires facilities to manipulate audit data for all systems at level C2 and above. Such facilities may include audit data analyzers and report generators.

20.3.5 *Device Labels*

20.3.5.1 *Definition.* The OSIF shall support the assignment of minimum and maximum security levels to all devices.

20.3.5.2 *Metric.* To be determined.

20.3.5.3 *Rationale.* The TCSEC requires that all devices have minimum and maximum security levels for all systems at level B2 or above. This requirement includes both physical devices (e.g., disks, terminals) and logical devices (e.g., interprocess communication).

20.3.6 *Basic DAC*

20.3.6.1 *Definition.* The OSIF shall support a mechanism for the enforcement of discretionary access control (DAC) based on users and groups.

20.3.6.2 *Metric.* To be determined.

20.3.6.3 *Rationale.* The TCSEC requires that minimal DAC facilities for level C2 include enforcement based on a user and group mechanism. or for levels B3 and above, the requirement is for access control lists (ACLs).

20.3.7 *DAC Inclusion/Exclusion*

20.3.7.1 *Definition.* The OSIF shall support the manipulation of access rights to specifically include or exclude access based on users or groups.

20.3.7.2 *Metric.* To be determined.

20.3.7.3 *Rationale.* The TCSEC requires that the DAC mechanism specifically allow for inclusion based on individual users or groups for levels C2 and above. For levels B3 and above, the TCSEC also requires that the DAC mechanism allow for exclusion based on individual users or groups.

20.3.8 *DAC Propagation*

20.3.8.1 *Definition.* The OSIF shall provide controls to limit propagation of access rights.

20.3.8.2 *Metric.* To be determined.

20.3.8.3 *Rationale.* The TCSEC requires that DAC rights granted to a user or group not be propagated to another user or group. In general, this is interpreted to mean that only the owner of an object (or a privileged user) may change the DAC on that object. This facility is required for security levels C2 and above.

20.3.9 *Labeling of Export Channels*

20.3.9.1 *Definition.* The OSIF shall support the restriction of the set of labels to be exported over each export channel.

20.3.9.2 *Metric.* To be determined.

20.3.9.3 *Rationale.* The TCSEC requires labeling of all imported and exported data. Further, it requires restrictions on labels exported over each export channel. It also requires that labels be retained with the data exported. This facility is required at systems rated B1 or above.

20.3.10 *Setting Communication Labels*

20.3.10.1 *Definition.* The OSIF shall support features to set or change each communication channel and I/O device to either single level or multilevel.

20.3.10.2 *Metric.* To be determined.

20.3.10.3 *Rationale.* The TCSEC requires that communications channels and I/O devices (e.g., tape and disk drives) be marked as single level or multilevel, and provide mechanisms to set or change that marking. This facility is required for systems rated B1 or above.

20.3.11 *Identification and Authentication*

20.3.11.1 *Definition.* The OSIF shall provide a protected mechanism to uniquely authenticate the identity of the user.

20.3.11.2 *Metric.* To be determined.

20.3.11.3 *Rationale.* The TCSEC requires that a sign-on procedure be used to uniquely identify users. The technology required is not specified, although it is typically a user ID and password. Storage of authentication data must be protected (e.g., protection of passwords). This facility is required for all secure systems.

20.3.12 *Labeling of Human Readable Output*

20.3.12.1 *Definition.* The OSIF shall support the marking of human readable sensitivity labels on all human readable output.

20.3.12.2 *Metric.* To be determined.

20.3.12.3 *Rationale.* The TCSEC requires labeling of all human readable output. This typically means labeling the top and bottom

of each page of hard copy output. This facility is required for all systems at levels B1 and above.

20.3.13 *Subject and Object Labeling*

20.3.13.1 *Definition.* The OSIF shall support labeling (i.e., setting and changing) of each subject and object.

20.3.13.2 *Metric.* To be determined.

20.3.13.3 *Rationale.* The TCSEC requires that each subject and object in the system be labeled. This facility is required of all systems at levels B1 and above.

20.3.14 *Label Contents*

20.3.14.1 *Definition.* The OSIF shall support the definition of a label for the system (i.e., classification, categories, markings, and special handling designators).

20.3.14.2 *Metric.* To be determined.

20.3.14.3 *Rationale.* The TCSEC requires labels contain classifications and categories at the B1 level and above. Markings and special handling requirements are not explicitly required by TCSEC.

20.3.15 *MAC Policy*

20.3.15.1 *Definition.* The OSIF shall support a security policy based on subject and object labels.

20.3.15.2 *Metric.* To be determined.

20.3.15.3 *Rationale.* The TCSEC requires that a policy exist and be enforced regarding access to subjects and objects based on a security policy. This requirement does not specify what the policy should be. However, the TCSEC specifies characteristics of the policy (e.g., no write-down, no read-up, use of classifications and categories).

20.3.16 *MAC Manipulations*

20.3.16.1 *Definition.* The OSIF shall support the manipulation of labels based on the security policy.

20.3.16.2 *Metric.* To be determined.

20.3.16.3 *Rationale.* The TCSEC requires that all manipulation of MAC labels be in accordance with the security policy for systems at level B1 or above.

20.3.17 *Object Reuse*

20.3.17.1 *Definition.* The OSIF shall provide that all objects are sanitized prior to allocation to a user.

20.3.17.2 *Metric.* To be determined.

20.3.17.3 *Rationale.* The TCSEC requires such sanitization of all objects in the system to prevent unauthorized disclosure of information. Note that "objects" in this case refer to hardware elements (e.g., registers and memory) in addition to traditional objects such as files and disk space. This facility is required for systems at levels C2 or above.

20.3.18 *User Notification of Sensitivity Label*

20.3.18.1 *Definition.* The OSIF shall support the prompt notification to a terminal user of each change in security level associated with that user during an interactive session.

20.3.18.2 *Metric.* To be determined.

20.3.18.3 *Rationale.* The TCSEC requires that the user be notified of such changes for levels B2 and above.

20.3.19 *Sensitivity Label Query*

20.3.19.1 *Definition.* The OSIF shall support the user's query of the subject's complete sensitivity label.

20.3.19.2 *Metric.* To be determined.

20.3.19.3 *Rationale.* The TCSEC requires that the user have access to the complete sensitivity label (including special handling requirements) for systems at level B2 and above.

20.3.20 *System Integrity*

20.3.20.1 *Definition.* The OSIF shall support features that can be used to periodically validate the correct operation of the hardware and firmware.

20.3.20.2 *Metric.* To be determined.

20.3.20.3 *Rationale.* The TCSEC requires that diagnostic and confidence tests be available to verify the correct functioning of the hardware at levels C1 and above.

20.3.21 *Identification of Users Based on Roles*

20.3.21.1 *Definition.* The OSIF shall support the identification of users based on roles.

20.3.21.2 *Metric.* To be determined.

20.3.21.3 *Rationale.* The TCSEC requires that various roles be provided with varying privileges. At a minimum, this separates the roles of operator, system administrator, and security administrator. This facility is required at levels B2 and above.

20.3.22 *Least Privilege*

20.3.22.1 *Definition.* The OSIF shall support the principle of least privilege.

20.3.22.2 *Metric.* To be determined.

20.3.22.3 *Rationale.* The TCSEC requires that least privilege be used at levels B2 and above. The exact privileges required are not defined here.

20.3.23 *Trusted Path*

20.3.23.1 *Definition.* The OSIF shall support a trusted communication path between the user and the system, activated exclusively by the user.

20.3.23.2 *Metric.* To be determined.

20.3.23.3 *Rationale.* The TCSEC requires that a trusted path be provided that provides for communication between the system and the user without the possibility of interception by any software other than the operating system. This facility is required at level B2 and above.

20.3.24 *Trusted Recovery*

20.3.24.1 *Definition.* The OSIF shall provide procedures or mechanisms or both to assure that after a discontinuity, recovery without a protection compromise is obtained.

20.3.24.2 *Metric.* To be determined.

20.3.24.3 *Rationale.* The TCSEC requires that systems at level B3 and above provide a trusted recovery mechanism. Such a mechanism is used for recovery to a sure state after a failure (e.g., a system crash) that left the system in an unknown security state.

20.4 DATA INTERCHANGE INTERFACES

20.4.1 *Data Interchange Services (Data Format Conversion)*

20.4.1.1 *Definition.* The OSIF shall support an access to services that perform data conversion (e.g., files, CPUs, or compilers).

20.4.1.2 *Metric.* To be determined.

20.4.1.3 *Rationale.* Operating systems must handle data from various sources. To properly transmit data from one source to the other, and to perform operations on data, the operating system needs a canonical data format that is unaffected by the external environment. The operating system shall provide service routines that support conversion of one data format to another for a "to be determined" set of processor internal data.

20.5 EVENT AND ERROR INTERFACES

20.5.1 *Event and Error Receipt*

20.5.1.1 *Definition.* The OSIF shall provide for the receipt and coordination of event and error information.

20.5.1.2 *Metric.* To be determined.

20.5.1.3 *Rationale.* Event and error information includes information available through such interfaces as to the hardware/firmware, the network, and the PSE as well as to the applications software. The fault information collection requirement (see section 20.11.1) also applies to the error information collection part of this requirement.

20.5.2 *Event and Error Distribution*

20.5.2.1 *Definition.* The OSIF shall provide for the distribution of event and error information.

20.5.2.2 *Metric.* To be determined.

20.5.2.3 *Rationale.* This requirement is a necessary corollary to the event and error receipt requirement (see section 20.5.1). The fault information request requirement (see section 20.11.2) also applies to the error information distribution part of this requirement.

20.5.3 *Event and Error Management*

20.5.3.1 *Definition.* The OSIF shall support the timely delivery of interrupt and other asynchronous events to system components and shall support the implementation of user-selectable error processing alternatives. Alternatives shall include, as a minimum, filtering, retry, ignore, and accumulate occurrences.

20.5.3.2 *Metric.* To be determined.

20.5.3.3 *Rationale.* For many applications, the OSIF must provide for the timely delivery of interrupt and other signal information to system components such as the operator and application software. This includes, particularly in

a tactical application, supporting the capability to display alerts to an operator at the system console. The fault detection thresholds requirement (see section 20.11.6) also applies to this requirement.

20.5.4 *Event Logging*

20.5.4.1 *Definition.* The OSIF shall support logging events to application-defined storage. The types of events and event sources shall be dynamically selectable/deselectable.

20.5.4.2 *Metric.* To be determined.

20.5.4.3 *Rationale.* Examples of event sources are specific processors or memory modules. By providing event and source selectability, this requirement provides for saving information that is more relevant to the application while conserving storage resources.

20.5.5 *Enable/Disable Interrupts*

20.5.5.1 *Definition.* The OSIF shall provide the ability to enable and disable interrupts.

20.5.5.2 *Metric.* To be determined.

20.5.5.3 *Rationale.* This requirement provides for interrupts as a whole to be turned on and off. The mask/unmask interrupts requirement, on the other hand, provides for individual interrupts to be made known/unknown.

20.5.6 *Mask/Unmask Interrupts*

20.5.6.1 *Definition.* The OSIF shall provide the ability to mask and unmask events.

20.5.6.2 *Metric.* To be determined.

20.5.6.3 *Rationale.* A system requires this capability during activities such as interrupt processing to lock out interrupts of a lower class from occurring or to mask out the interrupts from particular sources.

20.6 FILE INTERFACES

20.6.1 *Contiguous Read of a File*

20.6.1.1 *Definition.* The OSIF shall provide a capability to access information from an opened file in a contiguous stream.

20.6.1.2 *Metric.* To be determined.

20.6.1.3 *Rationale.* DBMS systems often read large blocks of data from the storage device to use as indices into other files. Such reads must be performed as quickly as possible. Limiting the number of seeks that a storage device must use (because of file fragmentation) can result in performance optimization. This function permits the access to a file that is contiguous on the storage medium.

20.6.2 *Protect an Area Within a File*

20.6.2.1 *Definition.* The OSIF shall provide a mechanism that restricts a file from access by requesters other than the requestor imposing the restriction.

20.6.2.2 *Metric.* To be determined.

20.6.2.3 *Rationale.* To protect simultaneous access and updates to a portion of files, the requestor of the access should be able to protect a section of the file until the user is finished updating. This is often called a file, byte, or record-locking capability.

20.6.3 *File Management Scheduling*

20.6.3.1 *Definition.* The OSIF shall support a capability to specify a response requirement for the service being requested for file management.

20.6.3.2 *Metric.* To be determined.

20.6.3.3 *Rationale.* For hard-deadline, real-time systems, file managers must schedule their service processing based on the response requirements of the requests submitted by the users. First in, first out (FIFO) scheduling is unacceptable for real-time applications. The file managers must also support the notion of preemption.

20.6.4 *File Management Suspend/Resume for Process*

20.6.4.1 *Definition.* The OSIF shall permit a requesting process to indicate whether it wishes to wait for completion of the requested service before continuing processing or to continue without waiting. In support of the latter instance, a means shall be provided to enable the requesting task to know the status of its service request or be notified of completion.

20.6.4.2 *Metric.* To be determined.

20.6.4.3 *Rationale.* In real-time applications, it is often necessary to request data and then to suspend processing until the data are available. It is also often the case that the task has no need to wait on the completion of the service before it continues processing. For example, if the request is posting data to a file, it is usually unnecessary, even undesirable, for the task to wait, especially in a distributed environment.

20.6.5 *File Management Block Requests*

20.6.5.1 *Definition.* The OSIF shall support the capability to update or retrieve a set of contiguous records in a file.

20.6.5.2 *Metric.* To be determined.

20.6.5.3 *Rationale.* Often in real-time systems, the position of a record in a file correlates to a time at which the information was derived. It is often necessary to read records that cover a particular span of time and to do so as expeditiously as possible. File managers should plan their accesses to minimize seek and latency when acting on these block requests.

20.6.6 *Round-Robin File Management*

20.6.6.1 *Definition.* The OSIF shall support a form of file access wherein, after a user-specified number of records have been written, new records will begin to overlay old records. Also, the OSIF shall support access requests for records that are based on relative position from newest/oldest record in file.

20.6.6.2 *Metric.* To be determined.

20.6.6.3 *Rationale.* Real-time systems keep historical files based on system requirements for coverage of some period of time. For example, 2 hours of

track history may be required online for rapid access. If a record is written to the file every 10 seconds, then the file will be 720 records long. The user wants to update the file such that the 721st record overlays the 1st record. The user should not have to manage the file pointers.

20.6.7 *Open a File*

20.6.7.1 *Definition.* The OSIF shall provide a capability to open a file for use.

20.6.7.2 *Metric.* To be determined.

20.6.7.3 *Rationale.* The user of a file must be able to indicate that the file is in use so that the operating system can maintain the necessary status and buffers. This function can also become a form of protection for file usage. A possible implementation (or requirement) would be that the file would be available to read, write, execution, or deletion to one, a group, or many users.

20.6.8 *Point Within a File*

20.6.8.1 *Definition.* The OSIF shall provide a capability to position the next access point within a file.

20.6.8.2 *Metric.* To be determined.

20.6.8.3 *Rationale.* Some method must be provided to allow random access to a file on storage medium.

20.6.9 *Read a File*

20.6.9.1 *Definition.* The OSIF shall provide a capability to access information from an opened file.

20.6.9.2 *Metric.* To be determined.

20.6.9.3 *Rationale.* Some method must be provided to access data from a file on a storage medium.

20.6.10 *Close a File*

20.6.10.1 *Definition.* The OSIF shall provide a capability to close a file that has been opened.

20.6.10.2 *Metric.* To be determined.

20.6.10.3 *Rationale.* Operating systems often restrict the number of files that can be open at one time. To release files that are no longer needed, the operating system user should be able to close a file that is opened.

20.6.11 *Delete a File*

20.6.11.1 *Definition.* The OSIF shall provide a capability to have a file physically removed from a storage medium.

20.6.11.2 *Metric.* To be determined.

20.6.11.3 *Rationale.* In addition to maintaining an organized storage medium and minimizing information redundancy, the operating systems need to provide a capability to delete files.

20.6.12 *Create a Directory*

20.6.12.1 *Definition.* The OSIF shall provide a capability to add a directory to the storage structure on the storage medium.

20.6.12.2 *Metric.* To be determined.

20.6.12.3 *Rationale.* To have a useful file system, there must be some mechanism to maintain separate user areas on the file storage medium. A possible implementation might be a hierarchical file system. Creating a directory enables an operating system user to add a user area.

20.6.13 *Specifying Default Directory*

20.6.13.1 *Definition.* The OSIF shall provide the capability to specify a processes' default directory.

20.6.13.2 *Metric.* To be determined.

20.6.13.3 *Rationale.* To have a useful file system, there must be some mechanism to maintain separate user areas on the file storage medium. A possible implementation might be a hierarchical system. Changing the current directory enables an operating system user to conveniently access a user area without providing path information.

20.6.14 *Delete a Directory*

20.6.14.1 *Definition.* The OSIF shall provide a capability to remove a directory from the storage structure on the storage medium.

20.6.14.2 *Metric.* To be determined.

20.6.14.3 *Rationale.* To have a useful file system, there must be some mechanism to maintain separate user areas on the file storage medium. A possible implementation might be a hierarchical system. Deleting a directory enables an operating system user to remove an unwanted user area.

20.6.15 *Shadow Files*

20.6.15.1 *Definition.* The OSIF shall support the creation, reading, writing, maintenance, and deletion of multiple, identical instantiations of a file. The multiple instantiations shall be viewed at the OSIF boundary as a single object.

20.6.15.2 *Metric.* To be determined.

20.6.15.3 *Rationale.* Applications will need a mechanism to maintain files at multiple locations or redundantly to satisfy performance or reliability requirements.

20.6.16 *Create a File*

20.6.16.1 *Definition.* The OSIF shall provide a capability to create a file.

20.6.16.2 *Metric.* To be determined.

20.6.16.3 *Rationale.* The user must be able to create a file, declaring attributes to be associated with the file, to be used for subsequent application specific storage/retrieval.

20.6.17 *Query File Attributes*

20.6.17.1 *Definition.* The OSIF shall provide a capability to query the attributes of a file.

20.6.17.2 *Metric.* To be determined.

20.6.17.3 *Rationale.* Each file has attributes associated with it (e.g., file size, creation date, modification date, owner, permissions, storage type, access privileges). The user must be able to query the attributes to become aware of the current state of the file. This is especially true for attributes that are dynamic and for attributes that are not under direct control of the user.

20.6.18 *Modify File Attributes*

20.6.18.1 *Definition.* The OSIF shall provide a capability to modify the attributes of a file.

20.6.18.2 *Metric.* To be determined.

20.6.18.3 *Rationale.* Each file has attributes associated with it (e.g., file size, creation date, modification date, owner, permissions, storage type, access privileges). The user must be able to modify the attributes to respond to changing mission capabilities.

20.6.19 *Write a File*

20.6.19.1 *Definition.* The OSIF shall provide a capability to write information to an opened file.

20.6.19.2 *Metric.* To be determined.

20.6.19.3 *Rationale.* Some method must be provided to write data to a file on a storage medium.

20.6.20 *Write Contiguous a File*

20.6.20.1 *Definition.* The OSIF shall provide a capability to write information to an open contiguous file.

20.6.20.2 *Metric.* To be determined.

20.6.20.3 *Rationale.* Some method must be provided to write data to a contiguous file on a storage medium.

20.7 GENERALIZED I/O INTERFACES

In the following requirements, the term "device" is used to indicate physical (i.e., a printer) or logical (i.e., pool of buffers) resources.

20.7.1 *Device-Driver Availability*

20.7.1.1 *Definition.* The OSIF shall provide the interfaces necessary to support the addition of device drivers.

20.7.1.2 *Metric.* To be determined.

20.7.1.3 *Rationale.* For an operating system to be expandable, a new device driver will have to be added. An explicit interface is needed to do this without having to go back to the vendor to incorporate this new driver.

20.7.2 *Open Device*

20.7.2.1 *Definition.* The OSIF shall provide the ability for a process to request the services of a particular device.

20.7.2.2 *Metric.* To be determined.

20.7.2.3 *Rationale.* An interface is needed to allow processes to request devices in the system and use the services of that particular device. In Ada terms, this interface would be used by packages such as TEXT_IO when doing an OPEN on a file. In SAFENET terms, this interface may be used by applications wishing to use the primitive SA_REGISTER req as outlined by the SAFENET standard. This primitive allows the user of SAFENET to receive a user service access point (USAP) identification, which is a logical device that only that

process or application can use. This gives it rights to the network.

20.7.3 *Close Device*

20.7.3.1 *Definition.* The OSIF shall provide the ability for a process to indicate that the services of a particular device, which had been previously allocated, are no longer needed.

20.7.3.2 *Metric.* To be determined.

20.7.3.3 *Rationale.* Once the device has been allocated and is no longer needed by a process, there needs to be a way to indicate to the operating system that this process is ready to release its control of the device. An example, in terms of Ada, of a process that would use this interface would be the TEXT_IO procedure CLOSE file. In SAFENET terms, this interface may be used by applications or processes wishing to use the primitive SA_CANCEL_req. This allows the application to give up its USAP identification, which is essentially saying it is giving up its control of the logical device (access to the network).

20.7.4 *Transmit Data*

20.7.4.1 *Definition.* The OSIF shall provide the ability to transfer specified block(s) of data to a device that has been previously opened by a process.

20.7.4.2 *Metric.* To be determined.

20.7.4.3 *Rationale.* An interface to allow a process to communicate with a device that it has already acquired is needed so that useful work can be done with this device. The TEXT_IO procedure PUT would be an example of a process that would use this interface. In SAFENET, this interface would allow applications wishing to use the service indicated by the primitive SA_SEND_req access to that service.

20.7.5 *Receive Data*

20.7.5.1 *Definition.* The OSIF shall provide the ability to receive data from a device that has been previously opened by a process.

20.7.5.2 *Metric.* To be determined.

20.7.5.3 *Rationale.* Processes will need an interface to use that will allow them to receive data from a device that has been previously assigned to it. In terms of Ada, the package TEXT_IO would need this interface for its procedure named GET. The SAFENET service, which is accessed by the primitive SA_REQUEST_req, could be accessed by applications through this interface to receive data from other nodes in the system.

20.7.6 *Device Event Notification*

Refer to requirements within section 20.5, "Event and Error Interfaces."

20.7.7 *Control Device*

20.7.7.1 *Definition.* The OSIF shall provide the mechanism to request a device to perform an action pertinent to the device.

20.7.7.2 *Metric.* To be determined.

20.7.7.3 *Rationale.* Processes need the capability to indicate some action to take place at a particular device through the OSIF. An example, using Ada, may be the procedure found in the package TEXT_IO called RESET file. This procedure may need some interface to the operating system to carry out the action requested. The SAFENET service, which is accessed by using the primitive SA_DISCONNECT_req, could possibly be accessed by applications through this interface. Example would be to sound an audible alarm, to abort an ongoing activity on a device and to initialize a device. Other examples may be sounding an alarm, aborting an ongoing activity on a device, etc.

20.7.8 *I/O Directory Services*

20.7.8.1 *Definition.* The interface shall support the use of directory services to map between logical names and physical devices or address and attributes of the devices.

20.7.8.2 *Metric.* To be determined.

20.7.8.3 *Rationale.* A service must be provided to keep track of all the peripheral devices and their attributes. This service may be centralized or distributed. Its existence implies a name registration function and authority to ensure the uniqueness of global names. The directory services function may be part of the basic operating system functions or it may be a part of the capabilities of a supporting subsystem such as SAFENET or a file management subsystem.

20.7.9 *Device Management Suspend/Resume for Processes*

20.7.9.1 *Definition.* The OSIF shall permit the requesting process to indicate whether it wishes to wait for completion of the requested service before continuing processing or to continue without waiting. In support of the latter instance, a means shall be provided to enable the requesting process to know the status of its service request.

20.7.9.2 *Metric.* To be determined.

20.7.9.3 *Rationale.* In real-time applications, it is often necessary to request data and then to suspend processing until the data are available. It is also often the case that the process has no need to wait for the completion of the service before it continues processing. For example, if the request is posting data to a device, it is usually unnecessary, even undesirable, for the process to wait, especially in a distributed environment.

20.7.10 *Mount/Dismount Device*

20.7.10.1 *Definition.* The OSIF shall support the capability to mount and dismount a logical or physical device.

20.7.10.2 *Metric.* To be determined.

20.7.10.3 *Rationale.* Of particular concern is the handling of traditional devices such as removable disk and tape storage entities. This mechanism may also be used to cause logical devices to become visible or invisible. Mount is defined as the action of mounting a logical or physical device that causes the entity to become a visible resource. This may be referenced by a device identifier and commonly by some logical name associated with a particular instantiation of media associated with the device. Associated with the mount capability is the implied ability to specify access rules to be applied to the mounted entity.

20.7.11 *Initialize/Purge Device*

20.7.11.1 *Definition.* The OSIF shall support device-dependent initialization and deinitialization (purge) functions for logical and physical devices.

20.7.11.2 *Metric.* To be determined.

20.7.11.3 *Rationale.* Disk and tape devices (and media) need to be formatted, erased, labeled, unlabeled, etc. Logical devices can apply these same functions to provide functions such as network connection initialization, etc.

20.8 NETWORK AND COMMUNICATIONS INTERFACES

In a system using components based on NGCR standards, there will frequently be a hierarchy of networked communication, data storage, and processing functions. At the base of this hierarchy may be a number of processing or storage units on a single board connected by an onboard bus. At the next level will be FUTUREBUS+ or non-NGCR backplane buses (e.g., VME). At the next level, there may be SAFENET, MIL-STD-1553B data buses, or non-NGCR defined LANs. At the highest level, but outside the scope of this set of requirements, there may be communications among systems on different Navy platforms.

In some application domains and for some application functions, the OSIF must provide explicit access to networked communication, data storage, and processing functions for both NGCR-defined communication components and similar non-NGCR-defined components. This is in addition to the implicit use of these capabilities implied in many other requirements.

Two processes make up a communications transaction regardless of their location. This includes either across a communications link or the two processes may possibly be residing on the same processor.

20.8.1 *Interface to and Control of Navy Standard Interprocessing Unit Buses*

20.8.1.1 *Definition.* The operating system shall provide explicit interfaces to and control of FUTUREBUS+, SAFENET, and MIL-STD-1553B in accordance with the standards or specifications defining each.

20.8.1.2 *Metric.* To be determined.

20.8.1.3 *Rationale.* These three sets of standards cover a broad range of capabilities. The OSIF, in addition to other functions, must provide the architecture, control, and management structure to integrate these components into a usable and functioning whole. The sets are defined as follows:

1. FUTUREBUS+ is an emerging IEEE set of standards for backplanes used to interconnect processing units and other boards within a card cage. Among other capabilities, it provides a precise, common time-of-day (TOD) clock and interprocess message passing for the FUTUREBUS+ interconnected devices.

2. SAFENET is the Navy's subset of ISO OSI standards. These are supplemented by additional specifications and implementation agreements drawn from the Manufacturing Automation Protocol (MAP 3.0) specification, Government

Open Systems Interconnection Profile (GOSIP, FIPSPUB 146), and SAFENET Working Group agreements. It provides services ranging from a variety of message communication services, to a file management and access system (FTAM), to support for the management of all components of the communication system. It also permits users to incorporate components based on ISO application layer standards not explicitly included in SAFENET.

3. MIL-STD-1553B is an older LAN with much lower performance characteristics than SAFENET. It only provides defined capabilities at the lower layers of the ISO/OSI model. However, because of cost considerations and familiarity with its capabilities in the air community, it may well continue to be used. The OS then must provide the architecture, control, and management structure to integrate this component into the total system.

20.8.2 *Interfaces to and Control of Other Network and Communication Entities*

20.8.2.1 *Definition.* The OSIF shall support explicit interfaces to the capabilities of multiple standard and proprietary backplane buses and LANs.

20.8.2.2 *Metric.* To be determined.

20.8.2.3 *Rationale.* The OSIF must be capable of interfacing to a variety of proprietary and standard LANs and backplanes to support program transportability. This is needed to enable it to be used with specialized systems that cannot economically be modified to use Navy standards. In the area of backplanes, VME and MULTIBUS are commonly used standards. Equipment such as DEC computers frequently use proprietary backplanes. In the area of LANs, the INTERNET standards (particularly TCP/IP) are frequently used.

20.8.3 *Reliable Virtual Circuit Communications*

20.8.3.1 *Definition.* The OSIF shall provide for the selection of reliable virtual circuit communications.

20.8.3.2 *Metric.* To be determined.

20.8.3.3 *Rationale.* Certain applications require the ability to transfer data between processes by means of a connection-oriented communications link. This link is established between processes and maintained for the transfer with error detection and correction support.

20.8.4 *Unreliable Virtual Circuit Communications*

20.8.4.1 *Definition.* The OSIF shall provide for the selection of unreliable virtual circuit communications.

20.8.4.2 *Metric.* To be determined.

20.8.4.3 *Rationale.* Certain applications call for the transfer of data over a connection-oriented link between processes but can withstand a certain amount of error better than the overhead associated with a reliable link. An example of such data is voice information and sensor data.

20.8.5 *Reliable Datagram Transfer*

20.8.5.1 *Definition.* The OSIF shall provide for the selection of reliable datagram transfer communications.

20.8.5.2 *Metric.* To be determined.

20.8.5.3 *Rationale.* Certain applications require the ability to transfer aperiodic information and do not require the establishment and maintenance associated with a connection-oriented transfer. Yet they require an acknowledgment that the information was successfully received at the destination.

20.8.6 *Unreliable Datagram Transfer*

20.8.6.1 *Definition.* The OSIF shall provide for the selection of unreliable datagram transfer.

20.8.6.2 *Metric.* To be determined.

20.8.6.3 *Rationale.* Certain applications require the ability to transfer information without the overhead associated with connection-oriented transfers and acknowledgments. These situations allow for the information to be sent with no assurance that it properly arrives.

20.8.7 *Request-Reply Service*

20.8.7.1 *Definition.* The OSIF shall support the ability to select request-reply communication services.

20.8.7.2 *Metric.* To be determined.

20.8.7.3 *Rationale.* Certain applications require communication services in the form of a request and a reply. In these situations, a requesting process sends the request and associated data to a service process. On receipt of the request, the service process performs requested service and returns the results to the requesting process.

20.8.8 *Unreliable Broadcast/Multicast Service*

20.8.8.1 *Definition.* The OSIF shall provide for the selection of an unreliable broadcast/multicast communication services.

20.8.8.2 *Metric.* To be determined.

20.8.8.3 *Rationale.* Certain applications require the ability to send a single message to all (broadcast) or several (multicast) destinations. In these situations, it is sometimes desirable not to have the overhead associated with connection-oriented transfers and reliable services. Those received will be checked for correctness and be discarded if not correct.

20.8.9 *Reliable Broadcast/Multicast Services*

20.8.9.1 *Definition.* The OSIF shall provide for the selection of reliable broadcast/multicast communication services.

20.8.9.2 *Metric.* To be determined.

20.8.9.3 *Rationale.* Certain applications require the ability to send a single message to all (broadcast) or several (multicast) destinations. In these situations, it is sometimes desirable to ensure the proper reception of the information at all or at some of the destinations.

20.8.10 *Atomic Broadcast/Multicast Services*

20.8.10.1 *Definition.* The OSIF shall provide for the selection of a reliable, atomic broadcast/multicast for communications services.

20.8.10.2 *Metric.* To be determined.

20.8.10.3 *Rationale.* Certain applications require the ability to send a single message to all (broadcast) or several (multicast) destinations. In these situations, synchronized behavior of the destinations is important so the communications subsystem must be able to guarantee that all destinations will receive the message within a stated time after the message is sent. By necessity, atomic messages must use a "reliable" broadcast/multicast service.

20.9 PROCESS MANAGEMENT INTERFACES

20.9.1 *Create Process*

20.9.1.1 *Definition.* The OSIF shall provide the ability to create processes with specified attributes.

20.9.1.2 *Metric.* To be determined.

20.9.1.3 *Rationale.* Processes and their environments need to be created prior to their execution. Attributes may include such things as process name, process priority, stack size, scheduling attributes, memory allocation, etc.

20.9.2 *Terminate Process*

20.9.2.1 *Definition.* The OSIF shall provide the ability to delete a process and recover all associated resources of that process.

20.9.2.2 *Metric.* To be determined.

20.9.2.3 *Rationale.* The OSIF must provide the service of deleting a process being executed. In addition, the operating system shall provide the ability to recover all the deleted processes' resources if so directed. In some cases, the responsibility for the return of the resource is that of the application.

20.9.3 *Start Process*

20.9.3.1 *Definition.* The OSIF shall provide a mechanism to designate a process as being ready to execute.

20.9.3.2 *Metric.* To be determined.

20.9.3.3 *Rationale.* The OSIF must provide the service of submitting a designated process to the processor's scheduling queue.

20.9.4 *Stop Process*

20.9.4.1 *Definition.* The OSIF shall provide the ability to make a process unavailable for scheduling.

20.9.4.2 *Metric.* To be determined.

20.9.4.3 *Rationale.* There are situations where processes are stopped from execution yet remain in a "wait state" where they may be restarted. Under these situations, the OSIF must maintain the process and its environment, yet not consider it for scheduling until specifically notified.

20.9.5 *Suspend Process*

20.9.5.1 *Definition.* The OSIF shall provide the ability for a process to suspend itself or another process from execution so that the suspended process retains resources, data, rights, and privileges, and execution may later be continued.

20.9.5.2 *Metric.* To be determined.

20.9.5.3 *Rationale.* The OSIF must provide the service of stopping a process from execution, yet maintain the process, the processes environment, and the processes data so that the process may be continued from the point of execution at which it was suspended.

20.9.6 *Resume Process*

20.9.6.1 *Definition.* The OSIF shall provide the ability to continue the execution of a process that has been previously suspended.

20.9.6.2 *Metric.* To be determined.

20.9.6.3 *Rationale.* The OSIF shall provide a service to allow a previously suspended process to continue execution from the point at which it was suspended.

20.9.7 *Delay Process*

20.9.7.1 *Definition.* The OSIF shall provide the ability to delay the scheduling of a process for a specified time period.

20.9.7.2 *Metric.* To be determined.

20.9.7.3 *Rationale.* This service allows for a process to be identified for scheduling prior to the actual time it is desired for the process to be scheduled.

20.9.8 *Interprocess Communication*

20.9.8.1 *Definition.* The OSIF shall provide the ability for processes to exchange information.

20.9.8.2 *Metric.* To be determined.

20.9.8.3 *Rationale* The OSIF must provide service(s) that allow processes to exchange data. These processes may or may not exist on the same processor. Examples of interprocess communication interfaces are shared files, lock files, shared memory, message passing, streams, pipes, FIFOs, signals, sockets, and access to higher level network services such as name servers and TCP/IP protocols.

20.9.9 *Examine Process Attributes*

20.9.9.1 *Definition.* The OSIF shall provide the ability for processes to examine the attributes of an existing process.

20.9.9.2 *Metric.* To be determined.

20.9.9.3 *Rationale.* Processes need the ability to read, analyze, and/or display the attributes of an existing process. Attributes may include such things as process name, process priority, stack size, scheduling attributes, memory allocation, etc.

20.9.10 *Modify Process Attributes*

20.9.10.1 *Definition.* The OSIF shall provide the ability for processes to modify the attributes of a particular process.

20.9.10.2 *Metric.* To be determined.

20.9.10.3 *Rationale.* Processes need the ability to modify the attributes assigned to a process when it was created when the significance of that process in the overall operation of the system changes. Examples of attributes that may require modification are process priority, stack size, scheduling attributes, memory allocation, etc.

20.9.11 *Examine Process Status*

20.9.11.1 *Definition.* The OSIF shall provide the ability for processes to examine the current status of a particular process.

20.9.11.2 *Metric.* To be determined.

20.9.11.3 *Rationale.* Processes need the ability to determine if another process has been created, started, deleted, stopped, suspended, etc.

20.9.12 *Process Identification*

20.9.12.1 *Definition.* The OSIF shall support the unambiguous identification of processes.

20.9.12.2 *Metric.* To be determined.

20.9.12.3 *Rationale.* Processes need the ability to identify other processes in the system in an unambiguous manner for such things as interprocess communication and examination of the status of other processes. This includes different processes within the system and multiple copies of a single process within the system.

20.9.13 *Save/Restart Process*

20.9.13.1 *Definition.* The OSIF shall support the ability for processes to be restarted from a saved state.

20.9.13.2 *Metric.* To be determined.

20.9.13.3 *Rationale.* The state of a process (as reflected in its execution status and local environment) is often the cumulative result of hours of running within a mission-critical system. It is commonly required in such systems to checkpoint the state of critical processes so that they may be restarted from a known good state if hardware or software faults are later detected. The checkpointed data would contain information relating to the processes status, environment, and state of its data.

20.9.14 *Program Management Function*

20.9.14.1 *Definition.* The OSIF shall provide multiprogramming support.

20.9.14.2 *Metric.* To be determined.

20.9.14.3 *Rationale.* This permits multiple Ada programs to be active simultaneously within a common processor. This requires the assignment of memory and processing resources to the programs.

20.10 **PSE INTERFACES**

20.10.1 *Debug Support*

20.10.1.1 *Definition.* The OSIF shall support the debugging of applications, specifically supporting the following capabilities:

1. Examine registers - a mechanism to examine registers of a selected resource in the system environment.

2. Alter registers - a mechanism to alter registers of a selected resource in the system environment.

3. Set/clear breakpoint - a mechanism to set/clear multiple breakpoints.

4. Set/clear watchpoints - a mechanism to set/clear multiple watchpoints.

5. Single-step execution - a mechanism to single step the execution of a software program.

6. Continue execution - a mechanism resume execution of a program after a breakpoint or watchpoint is encountered. The program shall resume execution at the next logical instruction.

7. Examine memory - a mechanism to read the contents of a process's address space.

8. Alter memory - a mechanism to modify the contents of a process' address space.

9. Query process environment - a mechanism to examine the state of a process.

10. Query Call Stack - The OSIF shall support the ability to determine the calling sequence of a process.

20.10.1.2 *Metric.* To be determined.

20.10.1.3 *Rationale.* The rationale for each of the ten required debug capabilities is as follows:

1. Examine registers - To fully access the state of the system, the programmer must be able to access CPU registers.

2. Alter registers - To control the state of the machine at a given point in execution, the user must be able to modify register values.

3. Set/clear breakpoint - Debugging tools require the ability to halt execution of the code at predetermined points to examine the state and status of the programming environment.

4. Set/clear watchpoints - Debugging tools require the ability to halt execution of the code when certain conditions or states occur to examine the state and status of the programming environment.

5. Single-step execution - Debugging tools require the ability to examine the state and status changes that occur when each line of code (instruction) is executed.

6. Continue execution - Debugging tools require the ability to resume normal execution of the program after it has been halted/stopped for examination of the programming environment.

7. Examine memory - Debugging tools requires the ability to examine the memory within the address space of a program.

8. Alter memory - Debugging tools requires the ability to modify the memory within the address space of a process.

9. Query process environment - Debugging tools require the ability to examine the state and status of the programming environment resulting from the execution of a process and does not exclude the states of associated queues and stacks (run, delay, and entry queues, etc.).

10. Query call stack - Debugging tools require the ability to examine the trail of calling sequences (e.g., providing the ability to determine "How did we get here?").

20.10.2 *Execution History*

20.10.2.1 *Definition.* The OSIF shall support the ability to monitor the execution history of a process, including information such as the following:

- Frequency of calls
- Length of calls
- Missed deadlines
- Length of queues
- Tasking of runtime systems (e.g., number of context switches, CPU time used)
- Dynamic paging activity
- Memory allocation (e.g., number of requests, block sizes, fragmentation, length of use)
- What OS services are being used (e.g., passing labels).

20.10.2.2 *Metric.* To be determined.

20.10.2.3 *Rationale.* For a performance monitor to create a history of events, the OS must provide the above information.

20.11 RELIABILITY, ADAPTABILITY, AND MAINTAINABILITY INTERFACES

20.11.1 *Fault Information Collection*

20.11.1.1 *Definition.* The OSIF shall provide for specifying the collection of available fault information.

20.11.1.2 *Metric.* To be determined.

20.11.1.3 *Rationale.* An application must be able to determine that a nonrecoverable fault has occurred, either by detecting the fault through

information available to it or by receiving some signal from other systems/ devices that a fault has been detected. This information is needed to increase the reliability of the system. This requirement provides a subset of the services needed (see section 20.5).

20.11.2 *Fault Information Request*

20.11.2.1 *Definition.* The OSIF shall provide for the receipt of fault information on request.

20.11.2.2 *Metric.* To be determined.

20.11.2.3 *Rationale.* The system must be able to determine that a nonrecoverable fault has occurred, either by detecting the fault through information available to it or by receiving some signal from other systems/devices that a fault has been detected. This information is needed to increase the reliability of the system. Receipt of fault information can be through active query or by use of a table that the application can access. This requirement provides a subset of the services required under event and error management (see section 20.5.3).

20.11.3 *Diagnostic Tests Request*

20.11.3.1 *Definition.* The OSIF shall provide for the initiation of diagnostic tests on specific request. The OSIF shall support initiation of diagnostic tests at specified intervals.

20.11.3.2 *Metric.* To be determined.

20.11.3.3 *Rationale.* Examples of these tests are built-in test equipment (BITE) tests, when software can initiate them, and firmware diagnostic tests of hardware components.

20.11.4 *Diagnostic Tests Results*

20.11.4.1 *Definition.* The OSIF shall provide the ability to determine the results of diagnostic tests.

20.11.4.2 *Metric.* To be determined.

20.11.4.3 *Rationale.* Receipt of the results of diagnostic tests can be through active query or by use of a table that the application can access. These diagnostic tests can be those initiated under the diagnostic tests request requirement (see section 20.11.3) or self-tests independently initiated by the effected system component.

20.11.5 *Operational Status*

20.11.5.1 *Definition.* The OSIF shall provide access to the operational status of all system components.

20.11.5.2 *Metric.* To be determined.

20.11.5.3 *Rationale.* System components include both software and hardware components such as buses, memory modules, processors, and I/O channels. Status indications include on, off, faulty, suspect, and the relief of a previously reported fault or overload condition.

20.11.6 *Fault Detection Thresholds*

20.11.6.1 *Definition.* The OSIF shall provide for specifying fault detection thresholds, which shall include, but not be limited to, the following:

1. Number of retry attempts, if applicable, that shall be made before an error is determined to be a nonrecoverable fault.

2. Maximum number of correctable errors that, if detected within a specified time, will classify the component as suspect or treat the collective errors as a nonrecoverable fault.

20.11.6.2 *Metric.* To be determined.

20.11.6.3 *Rationale.* The thresholds cited in the definition are required to detect intermittent faults. This requirement also applies to the event and error management requirement (see section 20.5.3).

20.11.7 *Fault Isolation*

20.11.7.1 *Definition.* The OSIF shall support the isolation of faults to a particular component.

20.11.7.2 *Metric.* To be determined.

20.11.7.3 *Rationale.* Not only must an OSIF provide detailed error information but it must also support localizing the fault so that applications software can be reconfigured and equipment repaired or replaced. Component, as used in the definition, refers to both hardware and software components.

20.11.8 *Fault Response*

20.11.8.1 *Definition.* The OSIF shall provide for the specification of actions to be taken on the occurrence of a fault. The OSIF shall support (at least) the following actions:

- Restart at a specified point for a specified fault
- Use of specified components as backup for faulty components
- Stop when a specified minimum set of components is no longer available
- Schedule of a specified process
- Report to another node.

20.11.8.2 *Metric.* To be determined.

20.11.8.3 *Rationale.* Navy applications, particularly those that are platform deployed, have traditionally required ever-increasing fault tolerant coverage. Part of that coverage has included providing a variety of fault responses to cover not only various kinds of faults but also various mission and processing requirements.

20.11.9 *Reconfiguration*

20.11.9.1 *Definition.* The OSIF shall support the dynamic reconfiguration of hardware and software.

20.11.9.2 *Metric.* To be determined.

20.11.9.3 *Rationale.* The set of available configurations for a particular implementation can be predetermined at system build time. These configurations will specify various configurations of the software to accommodate such variables as changes in mission requirements and operating in degraded modes. They will also specify the configurations that make sense for an implementation such as minimum memory requirements. The purpose of this requirement is to allow an implementation to make the best use of available hardware and software resources.

20.11.10 *Enable/Disable System Component*

20.11.10.1 *Definition.* The OSIF shall provide the ability to enable or disable a specified system component on request.

20.11.10.2 *Metric.* To be determined.

20.11.10.3 *Rationale.* This requirement supports reconfiguration. Examples of hardware components are processors, memory modules, and buses. Groups of software components could be subsystems or Ada programs.

20.11.11 *Performance Monitoring*

20.11.11.1 *Definition.* The OSIF shall support queries for snapshots of resource utilization and enabling or disabling monitoring of each resource.

20.11.11.2 *Metric.* To be determined.

20.11.11.3 *Rationale.* Snapshots are defined to be of a specified time exposure (as opposed to instantaneous).

20.11.12 *Set Resource Utilization Limits*

20.11.12.1 *Definition.* The OSIF shall support predefining and dynamically adjusting a process's utilization limits on a specified resource.

20.11.12.2 *Metric.* To be determined.

20.11.12.3 *Rationale.* Limits can be set at system build and then modified during runtime for each process and resource combination.

20.11.13 *Resource Utilization Limits Violation*

20.11.13.1 *Definition.* The OSIF shall support the detection and reporting of a process that violates its utilization limits for a resource.

20.11.13.2 *Metric.* To be determined.

20.11.13.3 *Rationale.* Once a limit is violated, the application may examine overall system performance and the situation to determine if a fault exists or if this load is consistent with operating demands. This requirement correlates directly to the set resource utilization limits requirement (see section 20.11.12).

20.11.14 *Checkpoint Data Structures*

20.11.14.1 *Definition.* The OSIF shall support the ability to replace specified existing data structures with those same structures as they appeared at a certain point in the past.

20.11.14.2 *Metric.* To be determined.

20.11.14.3 *Rationale.* Reconfiguration and diagnostics require the ability to move data structures in and out of memory. For example, in the event of the failure of a global memory module and consequent reconfiguration, applications could be warm-started in place, passing a pointer to the last checkpointed copies of mission-critical data structures.

20.12 RESOURCE MANAGEMENT INTERFACES

20.12.1 *Virtual Memory Support*

20.12.1.1 *Definition.* The OSIF shall support the selection of the virtual memory utilization parameters.

20.12.1.2 *Metric.* To be determined.

20.12.1.3 *Rationale.* On processor architectures supporting a larger virtual address space than the size of physical memory, the operating system implementation will generally support the virtual memory mapping hardware. The paging algorithm used and other virtual memory support parameters will need to be tailored to the application.

20.12.2 *Virtual Space Locking*

20.12.2.1 *Definition.* The OSIF shall provide the capability to lock certain application-specified regions of virtual code and data space into physical memory and for the subsequent release of such locks.

20.12.2.2 *Metric.* To be determined.

20.12.2.3 *Rationale.* For time-critical portions of applications, paging data and/or code to a secondary (mass) storage device would not allow for high-performance access. For fault tolerant applications, the fault-handling logic cannot be placed on a device that is likely to fail.

20.12.3 *Dynamic Memory Allocation and Deallocation*

20.12.3.1 *Definition.* The OSIF shall provide for allocation of a block of virtual or physical memory of the size specified and for deallocation of a previously allocated block.

20.12.3.2 *Metric.* To be determined.

20.12.3.3 *Rationale.* An application entity may require a global heap for its own dynamic memory management (e.g., the Ada runtime library), for dynamic load or relocation of code, for temporary buffers, etc. Such blocks, when no longer required by the application, should be re-entered into the pool of available physical memory.

20.12.4 *Dynamic Memory Protection*

20.12.4.1 *Definition.* The OSIF shall provide the ability to query and set memory-protection attributes.

20.12.4.2 *Metric.* To be determined.

20.12.4.3 *Rationale.* Mission-critical systems must guard against erroneous memory references (whether the result of software bugs, a security breach, or a hardware fault). While there is no foolproof approach to this, hardware memory protection provides a substantial level of confidence, but only if the OSIF provides for tailoring the memory protection to the application's needs. Any arbitrary block of memory may contain code, read/write data, read-only data, or (perhaps in multilevel secure systems) write-only data. Memory-protection requirements on a block may change over its lifetime.

Specification is required for all static code and data areas; any block of memory obtained through dynamic memory allocation may have its attributes specified during allocation. Memory-protection attributes for any (static or dynamic) block should be alterable at runtime. Protection violations should result in error events (see section 20.5.3).

20.12.5 *Shared Memory*

20.12.5.1 *Definition.* The OSIF shall support concurrent access, by several processes, to specified areas of physical memory, whether or not the involved processes exist on single or multiple processors.

20.12.5.2 *Metric.* To be determined.

20.12.5.3 *Rationale.* The concept of Ada library units requires shared memory for both code and data. Time-critical applications often cannot tolerate the overhead of message passing, rendezvous, or other forms of interprocess (or intertask) communication. Applications are responsible for sensible use of the shared memory resource (see section 20.13.2).

For virtual storage architectures, this will require a many-to-one mapping from virtual memory spaces to the shared physical page(s). Where the several processes are distributed across several processors separated by backplane or network interfaces, this will implicitly require interprocessor communication and synchronization.

20.12.6 *Allocate, Deallocate, Mount, and Dismount Services*

20.12.6.1 *Definition.* The OSIF shall support the allocation of devices to processes and subsequent deallocation of these devices. For devices with removable media, the OSIF shall also support mounting and dismounting of media.

20.12.6.2 *Metric.* To be determined.

20.12.6.3 *Rationale.* It is in the nature of some devices that they may be opened by several processes (i.e., shared), but many devices must be accessed exclusively by one process at a time. Some devices support opening of mountable volumes, and the OS should also provide explicit interfaces to specify the mounting and dismounting of such volumes. Control over such details is often left to ad-hoc interface-to-device drivers, but these common requirements are better handled by means of explicit application/OS/device-driver interfaces.

20.12.7 *Designate Control*

20.12.7.1 *Definition.* The OSIF shall provide the means to designate responsibility for maintaining the status and determining the configuration of a system resource.

20.12.7.2 *Metric.* To be determined.

20.12.7.3 *Rationale.* A basic purpose of an operating system is to regulate the control of system resources. This interface may be pre-runtime (static designation of control) or runtime (dynamic designation of control). The unit of software assuming the responsibility may be the operating system itself.

20.12.8 *Release Control*

20.12.8.1 *Definition.* The OSIF shall provide the means to release a previously assumed system resource status and configuration responsibility.

20.12.8.2 *Metric.* To be determined.

20.12.8.3 *Rationale.* Software must be able not only to assume responsibilities at runtime, but also to revoke and reassign them. This shall allow the operating system to designate responsibility for the system resource to another unit of software by use of the "designate control" interface.

20.12.9 *Allocate Resource*

20.12.9.1 *Definition.* The OSIF shall provide a means to designate particular process resources for use by a particular process.

20.12.9.2 *Metric.* To be determined.

20.12.9.3 *Rationale.* A basic purpose of an operating system is to regulate the control of system resources. The allocation request shall actually be honored by the entity currently designated as controlling the resource. Examples of units of system resources are an I/O channel, a block of physical memory, response to a specific class of hardware interrupt, a breakpoint register, a co-processor user identifier, and a connection over a LAN. The software making the allocation may be the operating system itself or may be application software assuming status and configuration responsibilities.

20.12.10 *Deallocate Resource*

20.12.10.1 *Definition.* The OSIF shall provide a means to relinquish particular system resources from a particular process.

20.12.10.2 *Metric.* To be determined.

20.12.10.3 *Rationale.* Software must be able not only to assume resources at runtime but also to revoke and reassign them.

20.12.11 *System Resource Requirements Specification*

20.12.11.1 *Definition.* The OSIF shall provide the ability to specify system resource requirements.

20.12.11.2 *Metric.* To be determined.

20.12.11.3 *Rationale.* The ability to modify the allocation of system resources based on operational needs is supported by this requirement. Specification of resource requirements before requesting resource allocation is required for effective management of resources, especially to prevent a deadlock among contenders for the resources.

20.12.12 *System Resource Capacity*

20.12.12.1 *Definition.* The OSIF shall provide a query of the storage or workload capacities of the system resources.

20.12.12.2 *Metric.* To be determined.

20.12.12.3 *Rationale.* The application (or entity controlling a resource) needs to know the availability and capacity of a resource to effectively allocate it during system operation.

20.13 SYNCHRONIZATION AND SCHEDULING INTERFACES

20.13.1 *Process Synchronization*

20.13.1.1 *Definition.* The OSIF shall provide an explicit mechanism by which two processes may synchronize their execution.

20.13.1.2 *Metric.* To be determined.

20.13.1.3 *Rationale.* Processes require the ability to synchronize their execution in real-time applications. To ensure predictable performance, this may include access to low-level synchronization mechanisms to ensure proper communication protocols. Synchronization should prohibit priority inversion situations.

20.13.2 *Mutual Exclusion*

20.13.2.1 *Definition.* The OSIF shall provide mutual exclusion and shall support mutual exclusion with timeouts.

20.13.2.2 *Metric.* To be determined.

20.13.2.3 *Rationale.* The system must have a low-level, mutual-exclusion mechanism available to all users and forbid priority inversions. If mutual exclusion is implemented with semaphores, then the semaphores must have operations available to create/destroy them, to claim/release them, and for priority queuing of processes waiting for a semaphore. Time-out mechanisms for processes waiting on semaphores must also be available. Additionally, developers must be able to query the status of a semaphore. All resources must have the ability to control critical sections for mutual exclusion. This is necessary for both safety and security. Processes that request a shared resource must have the ability/option to withdraw their request by means of a time-out, which may be zero, i.e., immediate withdrawal if the resource is not immediately available.

20.13.3 *Cumulative Execution Time of a Process*

20.13.3.1 *Definition.* The OSIF shall provide the ability to access the cumulative execution time of a process.

20.13.3.2 *Metric.* To be determined.

20.13.3.3 *Rationale.* When the scheduler is responsible for aperiodic process, it needs a means to determine the cumulative execution time so that priorities of the processes may be adjusted. The OSIF must provide a means by which applications software can monitor and establish the rules for scheduling and execution of aperiodic processes (see section 20.13.10).

20.13.4 *Attach a Process to an Event*

20.13.4.1 *Definition.* The OSIF shall support the ability to attach a process to an event.

20.13.4.2 *Metric.* To be determined.

20.13.4.3 *Rationale.* The application must be able to provide the scheduler with the information necessary to attach a process to an interrupt. This allows a process to respond to an external stimulus and helps to obtain more flexible scheduling. An example of an attached process is an event handler.

20.13.5 *Transaction Scheduling Information*

20.13.5.1 *Definition.* The OSIF shall provide the ability for a process to specify its response requirements for services.

20.13.5.2 *Metric.* To be determined.

20.13.5.3 *Rationale.* Scheduling in hard real-time systems must be done in a fashion to meet deadlines. For transactions that require a sequence of operations across a distributed system, the scheduling mechanisms involved require the ability of scheduling processes with respect to deadline requirements.

20.13.6 *Scheduling Delay*

20.13.6.1 *Definition.* The OSIF shall support the ability to delay the scheduling of a process.

20.13.6.2 *Metric.* To be determined.

20.13.6.3 *Rationale.* Refer to section 20.9.7, "Delay Process."

20.13.7 *Periodic Scheduling*

20.13.7.1 *Definition.* The OSIF shall provide for the periodic scheduling of a process.

20.13.7.2 *Metric.* To be determined.

20.13.7.3 *Rationale.* In real-time systems, certain process require the ability to be scheduled at a specific periodic rate. The rate may be specified with respect to a mean delta with a plus and minus limit of variance.

20.13.8 *Multiple Scheduling Policies*

20.13.8.1 *Definition.* The OSIF shall support multiple scheduling policies.

20.13.8.2 *Metric.* To be determined.

20.13.8.3 *Rationale.* Different applications require different scheduling algorithms.

20.13.9 *Selection of a Scheduling Policy*

20.13.9.1 *Definition.* The OSIF shall support the ability to select the scheduling policy to suit the need.

20.13.9.2 *Metric.* To be determined.

20.13.9.3 *Rationale.* It is perceived that once a scheduling algorithm is selected for an application, it remains static under normal conditions. However, mode changes or workload extremes may require dynamic alterations in scheduling policies. To meet this requirement, scheduling policies must be able to be altered without system reinitialization.

20.13.10 *Modification of Scheduling Parameters*

20.13.10.1 *Definition.* The OSIF shall support the ability to modify the values of the controllable scheduling parameters.

20.13.10.2 *Metric.* To be determined.

20.13.10.3 *Rationale.* Certain applications require the ability to dynamically modify the scheduling algorithms parameters used for selection of the process to be submitted for execution. The scheduler will need the freedom to change the priority of a process dynamically. As the system operates, different processes will assume prominent positions and, therefore, will require higher priorities. This adjustment of priorities must be dynamic to maximize system performance. The policies by which the priority adjustments are made must be controlled by the application software.

20.13.11 *Precise Scheduling (Jitter Management)*

20.13.11.1 *Definition.* The OSIF shall provide the ability for an application to indicate to the scheduler an exact specified time for starting a process.

20.13.11.2 *Metric.* To be determined.

20.13.11.3 *Rationale.* The scheduler must be able to guarantee that the process is executed at the exact time specified and is not unduly delayed. In real-time systems, completion of a scheduling event too early can be as bad as completion of a scheduling event too late, i.e., to miss a deadline. This phenomena is known as "jitter" and can cause performance problems in real-time systems. For real-time systems to perform as predicted and to ensure stability, a schedule must be met as closely as possible. It is not appropriate to complete an event early if it can be avoided. This is one of the things that separates real-time systems from time-sharing systems.

20.14 **SYSTEM INITIALIZATION AND REINITIALIZATION INTERFACES**

20.14.1 *Image Load*

20.14.1.1 *Definition.* The OSIF shall provide the capability to perform initial and reinitial executable image load (including data) both locally and remotely to and for each and all processor(s) throughout a system.

20.14.1.2 *Metric.* To be determined.

20.14.1.3 *Rationale.* The OSIF must support and provide the capability to load and reload initialize and reinitialize an executable image into each and all processor(s) throughout a system, both locally and remotely. This includes initial (cold start) and reinitial (cold, reconfigured (re)start and/or warm, reconfigured (re)start) of the operating system's designated processor and all others.

20.14.2 *System Initialization and Reinitialization*

20.14.2.1 *Definition.* The OSIF shall support the capability to initialize and reinitialize all system resources.

20.14.2.2 *Metric.* To be determined.

20.14.2.3 *Rationale.* A distributed, multiple-processor, real-time system must be initialized from a cold start and reinitialized after a cold start or warm (re)start so that the system configuration information necessary to execute the functions of the system is properly loaded in the different system components. This includes all communications, I/O ports, data storage and access components, etc. This means that the OSIF must support all necessary system initialization and reinitialization functions for a given application. This is not limited to image load, initialization, or reinitialization.

20.14.3 *Shutdown*

20.14.3.1 *Definition.* The OSIF shall provide the capability to perform planned, orderly shutdown at the local and remote levels for each and all processor(s) throughout a system.

20.14.3.2 *Metric.* To be determined.

20.14.3.3 *Rationale.* The OSIF must provide the capability to perform planned, orderly shutdown operations when required under crisis and noncrisis situations. This is a good resource management policy to attempt an orderly recovery in all situations up to the most catastrophic crash event.

20.15 TIME SERVICES INTERFACES

20.15.1 *Read Selected Clock*

20.15.1.1 *Definition.* The OSIF shall provide the ability to read selected clocks.

20.15.1.2 *Metric.* To be determined.

20.15.1.3 *Rationale.* The OSIF must have a facility for applications to read a selected clock, or set of clocks, in a system. Many applications have the need to time-stamp data either to coordinate events taking place in different parts of the system or to record when events take place so that data may be later properly processed or time ordered.

20.15.2 *Set Selected Clock*

20.15.2.1 *Definition.* The OSIF shall provide the ability to set selected clocks.

20.15.2.2 *Metric.* To be determined.

20.15.2.3 *Rationale.* The clocks used in a system may change over time. They may need to be set when a system component is initialized. The clock is a resource whose detailed management belongs to the OS. However, the setting of the clock and its coordination with external time sources are issues that must be left to the designer of a specific system. The OSIF, as the controller of common resources, must have a facility for applications to set a selected clock, or set of clocks, in a system.

20.15.3 *Synchronization of Selected Clocks*

20.15.3.1 *Definition.* The OSIF shall support the ability to selectively synchronize clock(s) in the system.

20.15.3.2 *Metric.* To be determined.

20.15.3.3 *Rationale.* The OSIF must have a facility for applications to selectively synchronize clocks in a system. The facility must allow synchronization of one clock, or set of clocks, to other clock sets in the system. These clock sets may be part of different subsystems, e.g., SAFENET, FUTUREBUS+, and a navigation system clock synchronized to Greenwich Mean Time. A means must exist to synchronize these to support those cases in which a common platform time base is required.

20.15.4 *Select a Primary Reference Clock*

20.15.4.1 *Definition.* The OSIF shall support the ability to select a primary reference clock for the system.

20.15.4.2 *Metric.* To be determined.

20.15.4.3 *Rationale.* The OSIF must have a facility for applications to select one primary reference clock or a set of primary clocks out of all clocks in a system and the set of systems integrated on a platform. This primary set must be able to be used to support clock synchronization throughout the system and also to support the selection of a backup reference clock in the event of failure of the primary. This is required to support those cases in which clocks of different quality are used in different subsystems or within a subsystem. A method must exist for indicating that a high-quality clock, or set of clocks, shall be used as the reference set to which others are synchronized.

20.15.5 *Locate the Primary Reference Clock*

20.15.5.1 *Definition.* The OSIF shall support the ability to locate the primary reference clock for a system.

20.15.5.2 *Metric.* To be determined.

20.15.5.3 *Rationale.* The OSIF must have a facility for applications to locate the primary reference clock in a system. This is required to support system management functions. For example, a failure in a system component could cause a system manager application to lose track of the identity of the current primary clock.

20.15.6 *Timer Services*

20.15.6.1 *Definition.* The OSIF shall support the setting and clearing of alarms and shall allow for notification at alarm time. The alarm time would be inclusive of either relative time difference or absolute time difference.

20.15.6.2 *Metric.* To be determined.

20.15.6.3 *Rationale.* The OSIF must have a facility for applications to set alarms for such things as watchdog timers, delays, etc. Once the time has expired, the notification of the alarm must be propagated to the appropriate recipient of the alarm. An Ada application and the Ada runtime system must be able to specify delays in terms of either an absolute or relative time basis. Some examples of these are as follows:

```
delay 2.0;                    -- delay for 2 seconds from now
delay_until(Next_Time);       -- delay until the absolute time (specified
                                 by the variable Next_Time)
```

20.15.7 *Precision Clock*

20.15.7.1 *Definition.* The OSIF shall provide a time resolution of one nanosecond to processes that is independent of the granularity of the underlying hardware.

20.15.7.2 *Metric.* To be determined.

20.15.7.3 *Rationale.* Applications must have a consistent, portable interface to the underlying clock(s) that allow them to use the full capability of the clock(s).

20.16 ADA LANGUAGE SUPPORT INTERFACES

20.16.1 *Create Task*

20.16.1.1 *Definition.* The OSIF shall support the capability to create an Ada task that supports the full set of Ada tasking operations as defined in the Ada LRM.

20.16.1.2 *Metric.* To be determined.

20.16.1.3 *Rationale.* An Ada runtime system must have the ability to create Ada tasks as logically concurrent threads of execution that are managed by the operating system. At the point of task creation, it must be possible to specify the Ada task's attributes (e.g., task name, priority, the task's master, stack space size, the number of entries) and the system resources (e.g., memory) needed to support the execution of the created Ada task (see section 20.9.1).

20.16.2 *Abort Task*

20.16.2.1 *Definition.* The OSIF shall support the capability to abort the execution of an Ada task as defined in the Ada LRM.

20.16.2.2 *Metric.* To be determined.

20.16.2.3 *Rationale.* An Ada runtime system must have the ability to abort Ada tasks and recover the resources previously held by those tasks (see sections 20.9.2 and 20.9.4).

20.16.3 *Suspend Task*

20.16.3.1 *Definition.* The OSIF shall support the capability to suspend the execution of an Ada task.

20.16.3.2 *Metric.* To be determined.

20.16.3.3 *Rationale.* An Ada runtime system must have the ability to suspend the execution of Ada tasks to support various task scheduling mechanisms (see section 20.9.5).

20.16.4 *Resume Task*

20.16.4.1 *Definition.* The OSIF shall support the capability to resume the execution of an Ada task.

20.16.4.2 *Metric.* To be determined.

20.16.4.3 *Rationale.* An Ada runtime system must have the ability to resume the execution of Ada tasks to support various task scheduling mechanisms (see section 20.9.6).

20.16.5 *Terminate Task*

20.16.5.1 *Definition.* The OSIF shall support the capability to terminate the execution of an Ada task as defined in the Ada LRM.

20.16.5.2 *Metric.* To be determined.

20.16.5.3 *Rationale.* An Ada runtime system must have the ability to terminate the execution of Ada tasks to support the full semantics of the Ada tasking model.

20.16.6 *Restart Task*

20.16.6.1 *Definition.* The OSIF shall support the capability to restart the execution of an Ada task at a point immediately following its elaboration code.

20.16.6.2 *Metric.* To be determined.

20.16.6.3 *Rationale.* An Ada runtime system must have the ability to restart the execution of Ada tasks to support mode change operations (see section 20.9.13).

20.16.7 *Task Entry Calls*

20.16.7.1 *Definition.* The OSIF shall support simple, timed, and conditional Ada task entry calls as defined in the Ada LRM.

20.16.7.2 *Metric.* To be determined.

20.16.7.3 *Rationale.* An Ada runtime system must have the ability to implement all forms of Ada task entry calls, namely simple, timed, and conditional calls.

20.16.8 *Task Call Accepting/Selecting*

20.16.8.1 *Definition.* The OSIF shall support the various forms of accepting Ada task entry calls as defined in the Ada LRM. In particular, the OSIF shall support simple accepts, simple selective waits, selective waits with delay alternatives, selective waits with an else clause, and selective waits with a terminate alternative.

20.16.8.2 *Metric.* To be determined.

20.16.8.3 *Rationale.* An Ada runtime system must have the ability to implement all forms of accepting and selecting Ada task entry calls as defined in the Ada LRM.

20.16.9 *Access Task Characteristics*

20.16.9.1 *Definition.* The OSIF shall support the capability to access an Ada task's attributes and characteristics.

20.16.9.2 *Metric.* To be determined.

20.16.9.3 *Rationale.* An Ada runtime system must have the ability to read a task's attributes (e.g., task ID, execution state, available CPU time compared with specified time budget) and, also, to read and write a task's characteristics (e.g., priority, period, phase) to implement various scheduling mechanisms (see section 20.9.10).

20.16.10 *Monitor Task's Execution Status*

20.16.10.1 *Definition.* The OSIF shall support the ability to monitor a task's execution status, in particular, the amount of accumulated CPU time that has been used by the task.

20.16.10.2 *Metric.* To be determined.

20.16.10.3 *Rationale.* An Ada runtime system must have the ability to monitor a task's execution behavior in terms of the amount of accumulated CPU time that it has used. Such information is needed on a task-by-task basis to implement certain real-time scheduling algorithms (e.g., deferrable server, sporadic server, degraded mode for imprecise results) (see section 20.9.11).

20.16.11 *Access to a Precise Real-Time Clock*

20.16.11.1 *Definition.* The OSIF shall support access (e.g., read/write, setting alarms) to a precise, continuous real-time clock.

20.16.11.2 *Metric.* To be determined.

20.16.11.3 *Rationale.* An Ada runtime system must have the ability to read from and write to a precise real-time clock. Also, the Ada runtime must be able to set or remove timer alarms to be triggered at a specified time in the future, to implement Ada's delay statement and timed entry calls. Setting these timer alarms is also useful for implementing precise, periodic scheduling of Ada tasks; watchdog timers; and timeouts on communication primitives (see sections 20.15.6 and 20.15.7).

20.16.12 *Access to a TOD Clock*

20.16.12.1 *Definition.* The OSIF shall support read and write access to a TOD clock.

20.16.12.2 *Metric.* To be determined.

20.16.12.3 *Rationale.* An Ada runtime system must have the ability to read from and write to a TOD clock to support the operations defined in package calendar. Furthermore, an Ada application program must be able to read the TOD (through a calendar.Clock function call) clock to effect a delay until a specified time in the future (e.g., delay (Next_Start_Time - calendar.Clock)).

20.16.13 *Dynamic Task Priorities*

20.16.13.1 *Definition.* The OSIF shall support the capability to get and set the execution priority of an Ada task.

20.16.13.2 *Metric.* To be determined.

20.16.13.3 *Rationale.* An Ada runtime system must have the ability to dynamically control the execution priority of an Ada task to implement various scheduling mechanisms (see section 20.13.10).

20.16.14 *Scheduling Policy Selection*

20.16.14.1 *Definition.* The OSIF shall support the capability to get and set the policy that is to be used to schedule Ada tasks.

20.16.14.2 *Metric.* To be determined.

20.16.14.3 *Rationale.* An application must be able to select the scheduling policy (e.g., priority preemptive, time slicing within equal priority levels) that will be used by the operating system to schedule executing tasks. The open issues include: (1) one versus multiple policies in effect at a given time; (2) how various policies interact, and (3) the scope of a scheduling policy (e.g., entry queues, run queue).

20.16.15 *Memory Allocation and Deallocation*

20.16.15.1 *Definition.* The OSIF shall support the capability to create and/or delete a pool of memory that can be used as a heap for allocation and deallocation of smaller access collections. Furthermore, the OSIF shall support the capability to allocate data objects from both an independently allocated heap (e.g., Ada access collection) and a global pool of unallocated memory. It must be possible for the application to notify the operating system when use of the heap space is no longer required.

20.16.15.2 *Metric.* To be determined.

20.16.15.3 *Rationale.* An Ada runtime system must have the ability to allocate memory space for access variables (i.e., access collections) and task stacks. Heap management is necessary to prevent memory fragmentation and other garbage collection-related problems, and to allocate and deallocate large chunks of memory based on dynamic scope (see section 20.12.3).

20.16.16 *Interrupt Binding*

20.16.16.1 *Definition.* The OSIF shall support the capability to bind and unbind an interrupt to Ada application code, in particular, to at least an Ada interrupt task entry.

20.16.16.2 *Metric.* To be determined.

20.16.16.3 *Rationale.* An Ada runtime system must have the ability to attach and detach code to a device interrupt. The Ada LRM suggests that interrupts can be bound to task entries using an address clause. Moreover, a conventional interrupt service routine (ISR) approach requires that the ISR code be directly tied to a device interrupt (see section 20.13.4).

20.16.17 *Enable/Disable Interrupts*

20.16.17.1 *Definition.* The OSIF shall support the capability to enable and disable interrupts.

20.16.17.2 *Metric.* To be determined.

20.16.17.3 *Rationale.* Ada applications and the Ada runtime system must have the ability to control interrupts by enabling and disabling them. Often times, controlling interrupts is used as a programming technique for implementing critical sections of code. Disabling and enabling interrupts is also necessary for controlling a device's operations.

20.16.18 *Mask/Unmask Interrupts*

20.16.18.1 *Definition.* The OSIF shall support the capability to mask and unmask device interrupts.

20.16.18.2 *Metric.* To be determined.

20.16.18.3 *Rationale.* Ada applications and the Ada runtime system must have the ability to control device interrupts by masking and unmasking them. Masking and unmasking interrupts is also necessary for controlling a device's operations.

20.16.19 *Raise Exception*

20.16.19.1 *Definition.* The OSIF shall support the capability to raise an exception in an Ada task.

20.16.19.2 *Metric.* To be determined.

20.16.19.3 *Rationale.* An Ada runtime system must have the ability to raise an exception in any given Ada task. In particular, an Ada runtime system must be able to raise an exception in a task when hardware-detected exceptions (e.g., overflow, access violation) occur.

20.16.20 *I/O Support*

20.16.20.1 *Definition.* The OSIF shall support for Ada input/output as described in chapter 14 of the Ada LRM.

20.16.20.2 *Metric.* To be determined.

20.16.20.3 *Rationale.* In conformance to the Ada LRM, the correspondence between the input/output supported for Ada and all other input/output supported by the interface must be clearly defined. The interface must provide access from Ada to files written by other languages (if any).