(2)

Q14 - Interface Standards
Informal Technical Data
Ada Interfaces to POSIX

STARS-QS-02021/005/00
14 April 1989

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 April 1990	Final

4. TITLE AND SUBTITLE
Interface Standards Informal Technical Data,
Ada Interfaces to POSIX

5. FUNDING NUMBERS
STARS Contract
F19628-88-D-0031

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER
GR-7670-1070(NP)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
Headquarters, Electronic Systems Division (AFSC)
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Q14-02021/005/00

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report considers the applicability of the Portable Operating System Interface (POSIX) to the development of the Software Engineering Environment (SEE) for the Software Technology for Adaptable, Reliable Systems (STARS) program. It compares and contrasts characteristics and potential overlap of object management interfaces (in this example, CAIS-A) and POSIX. Because there are overlaps in the objectives for POSIX and an object manager, there are some apparent overlaps in the functions provided by the interfaces. It is concluded that there is no actual overlap in function in the I/O model for persistent data nor in the execution control models. Interfaces like CAIS-A should be the portable interface set for tools, while POSIX offers a method for gaining portability for the CAIS and other interfaces in the base level of the SEE.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Portable Operating System Interface (POSIX) Ada Binding	16
	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

STARS-QS-02021/005/00

INFORMAL TECHNICAL DATA

STARS Q14 INTERFACE STANDARDS

Ada INTERFACES TO POSIX

CONTRACT NO. F19628-88-D-0031

SDRL Q14-02021

14 April 1989

PUBLICATION NO. GR-7670-1070 (NP)

Prepared for:
Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

PREFACE

This document was produced by Unisys Corporation, Defense Systems, in support of the Unisys STARS Prime contract. This SDRL is for the Interface Standards Task, Q14, of the Unisys STARS First Increment. It is CDRL type A005, SDRL number 02021, Volume 5, for the Ada Interfaces to POSIX.

1. Executive Summary

POSIX is a standard interface set for operating systems designed for portability. POSIX can provide the portability layer underlying the Software Engineering Environment (SEE) to allow the baseline environment to be easily ported. It is expected that a wide range of COTS implementations, spanning a large range of hardware architectures and operating systems, will be available.

CAIS-A [CAIS-A88] has been proposed as the primary set of interfaces for the SEE object base and also for establishing portability of the software within the SEE. Because there are overlaps in the objectives for POSIX and the CAIS, there are some apparent overlaps in the functions provided by the interfaces. The main difference between the two sets of interfaces is that the CAIS contains a large project database which is responsible for organizing resources in a common manner. It is the existence and architecture of this object model which makes it desirable to have Input-Output and execution control interfaces as part of CAIS-A. Thus, it is concluded that there is no actual overlap in function in the Input-Output model for persistent data nor in the execution control models. The CAIS-A interfaces should be the portable interface set for tools, while POSIX offers a method for gaining portability for the CAIS and other interfaces in the base level of the SEE.

As a result of its importance to the SEE, it is recommended that the POSIX standardization effort be supported by STARS and that the STARS involvement in standardization efforts now underway continue.

2. Introduction

The Portable Operating System Interface (POSIX[1]) is not itself an operating system, but is a set of interfaces for accessing operating system functions. This subtle distinction is important, for while the interface set has its historical roots in UNIX[2], and is very similar to the System V Interface Definition (SVID), the POSIX interfaces may be implemented for access to nearly any existing operating system, including the various forms of UNIX, and the proprietary operating systems developed by such manufacturers as Digital Equipment Corporation, Unisys and IBM, and the multitasking operating systems running on personal computers. This expectation of widespread

---

[1]POSIX is pronounced pahz-icks, as in positive.
[2]UNIX is a registered trademark of AT&T.

applicability is encouraged by the fact that nearly all operating system vendors are represented on the POSIX committee, including Digital Equipment Corporation, IBM, Unisys, AT&T, Sun, Convergent Technologies, and Microsoft. Industry organizations such as X/Open and /usr/group are also represented on the committee.

Such support to the standard leads to expectation of widespread availability of implementations. Many vendors, including AT&T, have indicated intent to provide POSIX conforming interface sets, and a few, including IBM, already claim conformance to the standard. With such wide availability of Commercial-Off-The-Shelf (COTS) implementations available, it makes sense to try to exploit POSIX as a portability vehicle for the Software Engineering Environment (SEE). Since the Unisys Baseline Environment task, Q8, has already ported an implementation of CAIS-A to UNIX, the POSIX interface set is substantially validated as a potential portability base.

The significance of POSIX to the STARS program is presented in the following sections which describe the motivation for why POSIX is an important standard to STARS, how POSIX relates to the STARS SEE, the standards process surrounding the standards development, especially that of the associated Ada standard, and finally, recommendations concerning the role of POSIX in the STARS program are presented.

2.1. Motivation: A Historical Perspective

There were several factors contributing to the need for a standardized interface to operating systems. First and foremost there is the problem that nearly all operating systems are developed in a proprietary manner. All major manufacturers have developed proprietary operating system designs. For example, Unisys has A Series MCP and 1100 OS, IBM has a wide variety of operating system styles and designations, and Digital has VAX/VMS. There are some software companies which offer their own operating systems: Pick, with the Pick Operating System, and Microsoft with its MS DOS and OS2 operating systems. The most widespread operating system available is UNIX, with its multitude of incompatible variations: AT&T's System V, Berkley's BSD, various incompatible real-time extensions of UNIX such as the one produced by Interactive Systems, and DARPA's MACH, an experimental operating system developed by CMU.

With such a wide variety of operating systems, application code is difficult to produce which is truly portable. For example, such a simple action as gaining access to a sequential file is made extremely difficult because each proprietary system defines different rules for creation of

directory structures, lengths of file names, syntax for specification of a file name, and rules for creation of and access to files. Other operating system dependencies arise in many other areas, particularly in multitasking and interprocess communication. These problems are visible to the application programmer, even in such a language as rigidly standardized as Ada. Thus, it is clear that a well-defined interface to operating systems would help with program portability.

In an attempt to get standardization on an operating system interface set, basing it on UNIX seems a natural choice because it is already available on a wide range of systems from microprocessor systems to mainframes. However, no particular proprietary variety of UNIX can be chosen as the standard because of the chilling effect such a choice has on competition: as the chosen developer continued to improve the interface set, competing developers would be later to arrive on the market, and would not be likely to be as competitive.

In early attempts to gain standardization for government procurements, operating systems were required to be in conformance to the AT&T System V Interface Definition (SVID), a public domain interface definition. However, such specification still runs against competitive notions because the validation suite for SVID is a proprietary product of AT&T, and the interface definition is under no formal control: AT&T may change it at any time, giving it an unfair advantage. Also, UNIX is an old system, which was developed with a minimum of requirements. Thus, it is missing many features, such as an Ada binding and support for real time systems and supercomputing. Such necessary extensions to the interface set remain unspecified and are developed by a wide range of vendors in incompatible ways. Therefore, a formally controlled standard with a well established process for continued improvement and evolution is necessary.

2.2. Purpose of POSIX

POSIX is intended to be a portable operating system interface which promotes

- program portability through standardized programatic interfaces,
- consistent interpretation across languages through a single, functional specification, and
- reduced programmer training through consistent user interfaces.

The interface set is also expected to evolve and expand. This is particularly important because the UNIX operating system model is very old (1960's vintage), and was not developed to handle a wide variety of application areas. It was primarily developed as a simple development environment for use by the developers at AT&T's Bell Laboratories. As a result, the model is constantly criticized for its lack of support for such facilities as inexpensive multitasking ("lightweight processes"), transaction processing and multiprocessors.

In addition, the interface set must evolve to eliminate its strong dependence on the C language. This dependence reveals itself in many ways, including its use of signals and its lack of memory management facilities. In both examples, the interface set relies heavily on C language features (procedure parameters and the malloc function) which do not exist in Ada, nor in many other languages.

2.3. The Importance of POSIX to STARS

STARS fully recognizes the problems of portability without a well defined set of interfaces to the operating system on which the SEE is hosted. One of the primary goals in SEE development is the ability to port the SEE to a wide variety of systems quickly and inexpensively, while preserving consistent functioning of the environment.

The easiest way to do that is to host the baseline environment (the object management system, user interface management system, and the portability interfaces) on a set of interfaces which are supported by a wide variety of vendors. If the interfaces are available as commercial off-the-shelf products, then rehosting costs will be minimal and STARS can concentrate its efforts on its more important tasks.

As previously noted, POSIX is expected to be available on a wide variety of systems ranging from microprocessor based systems to mainframes. In fact, there is already a commercially available validation suite and vendors who have verified conformance to the standard.

The Ada binding is receiving reasonably good support from the community, including commercial vendors. At least three prototyping efforts have begun experimentally, including the public domain effort within the Engineering Information System (EIS) program. The EIS program is a tri-service program, initiated by the VHSIC program, and managed by AFWAL (Air Force Systems Command). Unfortunately, the EIS efforts towards an Ada/POSIX binding have been halted temporarily, but are expected to resume.

All of these factors contribute to the firm belief that the Ada interfaces will be available in experimental form in the public domain soon, and in a more robust form from commercial vendors soon after standardization.

POSIX provides a substantial base on which the DoD standard Ada Project Support Environment Interface Sets (APSE[3]), CAIS and CAIS-A, can be hosted. Thus, it is recommended that STARS continue to be supportive of, and involved in, the standardization of the Ada POSIX interfaces.

3. CAIS-A and POSIX

The objectives of CAIS (and CAIS-A) [RAC86] and POSIX have considerable overlap. One of the primary CAIS objectives is the ability to support Ada program execution on a wide variety of computers with a wide variety of operating systems, as well as on bare machine, with no supporting operating system. POSIX objectives are similar in that it specifies an operating system interface which can be supported by a wide variety of operating systems.

This overlap in objectives causes a wide overlap in function between the two interface sets. The overlaps are most noticeable in the two areas of program execution control (processes) and Input-Output. In any case of overlap between available standard interface sets a decision must be made as to which interfaces should be visible in the environment. The following sections describe the overlap in function and provide recommendations on how STARS should proceed.

3.1. Primary CAIS-A and POSIX Differences

CAIS-A and POSIX, while sharing many similar objectives, also differ in one substantial way: CAIS-A, as its name suggests, is intended as a modern project support environment for Ada program development and execution. POSIX is both more limited and broader in scope. It is not a project support environment and it tries to address program execution in a language independent manner.

The primary way in which these differences are manifested are in the existence of the CAIS-A node model. CAIS-A

[3]The term APSE was originally coined to be Ada Programming Support Environment. However, the scope expected to be covered by a complete APSE now encompasses all aspects of project development, making the term "Project" more appropriate.

supports a project database which is used by the various tools to share a common view of the project resources across the entire SEE. POSIX has no formalized communication technique between tools. Tools either cooperate by informal agreement as to which files are to be maintained in common, or through streams of ASCII text with the pipe construct. In either case, POSIX data structuring is entirely by informal agreement between the tools with no verification of the data storage structure. CAIS-A has a formal definition encoded as part of the project database and data structure definitions are shared between tools, allowing for validation of the structure on storage and retrieval.

Another main difference is in access control. The CAIS-A model provides interfaces to allow tools to operate within a Trusted Computer System (TCS) which meets B3 criteria. When implemented in a TCS, CAIS supports operations in a multi-level secure environment.

3.2. Input-Output

As a primitive operating system function, Input-Output exists in both interface sets. If one assumes that POSIX is an appropriate portability vehicle for the baseline environment and that the CAIS interface set provides much of the functional base for the environment, then the question arises of which I/O set is to be used by the tools in the environment.

The existence of two I/O systems does not actually raise a conflict for access to persistent data. The CAIS I/O is present and required for access to the contents of the nodes in the project database. Since there is no POSIX equivalent to the database, the use of POSIX I/O is not an option. The role of POSIX I/O is as an implementation vehicle for the CAIS node model.

There is, however, a conflict in the I/O interfaces to be used for non-persistent I/O. Since Terminal I/O is not heavily reliant on the node model for its functioning, there is no strong reason for user interaction to be done through the CAIS I/O functions. Additionally, the user interface functions provided by both CAIS-A and POSIX are very primitive, offering only simple interface styles without support for such devices as bit-mapped displays and pointing devices. Since it has been shown that these more modern devices provide a substantial improvement in programmer productivity, the SEE should provide a user interface vehicle which will support these devices as well as today's common text-oriented devices.

In conclusion, it is recommended that CAIS-A I/O be used for access to the contents of nodes in the project database and a more modern User Interface Management System should be used for the human-machine interface. POSIX can be used to provide the low level device support required for implementation of the higher level UIMS, but such implementation details are not visible to the user.

3.3. Program Execution Control

Another apparent conflict arising from the overlapping objectives is in program execution control (tasks and process management). Both, CAIS-A and POSIX, provide processes for control of program execution. In addition, Ada provides tasks for many of the same functions, adding additional choices for program execution control.

In dealing with program execution that can be managed through the use of Ada tasks, then clearly one would stay within the language rather than go to a non-Ada execution model. The primary case in which one must leave the Ada execution model is to execute another Ada program, such as a tool in the SEE. This is especially necessary for certain tools in the environment, such as the Ada Command Environment (ACE).

While there is some overlap in function in the program execution control, the process model in CAIS-A is intimately tied to the node model. It is through the process model that access control for use in trusted systems is established, database views are defined, and it is the mechanism which provides for the database transaction model, especially multiprocessor transactions. Thus, the process model cannot be replaced by the more primative model present in POSIX.

3.4. The Role of POSIX in the STARS Architecture

The use of CAIS-A as the baseline environment, coupled with a mature User Interface Management System, provides a completely portable environment on which the tools of the SEE may be hosted. Since this layer provides a complete set of functions required for the SEE, and is also a portable interface layer, one must question the importance of an older operating system model such as POSIX to STARS.

It is clear that if one of the interface sets provides a sufficient set of functions for implementation of the tool set, then the other interface set should not be visible. One of the goals of the SEE architecture is to minimize the number of interfaces in order to reduce the number of decisions that need to be made (and possibly re-made) by a

tool developer and also to reduce the cost in construction of the interfaces.

POSIX provides a good set of fundamental interfaces between an implementation of the baseline environment and the underlying machine and operating system which can be of use in meeting the portability objectives of the SEE. Through the low level POSIX primitives for program execution control and Input-Output, one may construct a baseline environment in which the CAIS-A implementation consists mainly of support for the project database. Implementation of the program execution control and I/O facilities in CAIS-A become a matter of implementing the project database manipulation logic and then use of the POSIX mechanisms for the low level control.

Viewing POSIX as an implementation vehicle for the baseline environment, means that POSIX is not a visible part of the SEE architecture. It is strictly an engineering detail in implementation of the SEE. What makes this particular engineering decision one that is architecturally important is the fact that using it as a portability layer between the project database/user interface layer and the hardware/operating system layer provides the benefits of portability: reduced cost of porting the baseline environment between systems. This supports the software first goal of developing a hardware and operating system independent SEE.

4. POSIX Standards Activity

POSIX standards activities are described in the following sections with a description of the overall organization of the committee, the current state of standardization, the approach being taken to POSIX standardization, and then an in-depth look at the Ada standardization process. The Ada standards process is described in terms of the philosophy taken towards the binding, with a comparison to the philosophies used on other Ada bindings, and, finally, the Ada binding development plan and progress.

4.1. POSIX Committee Organization

The POSIX standardization effort has been initiated under the auspices of the IEEE Technical Committee on Operating Systems (TCOS). The committee is called 1003 and is broken into several formally identified subcommittees, each with its own charter. The current list of committees is:

Committee	Subject
1003.0	Guide to POSIX Applications Portability
1003.1	System Interfaces
1003.2	Shell and Utilities
1003.3	Testing and Certification
1003.4	Real Time
1003.5	Ada Bindings
1003.6	Security
1003.7	System Administration
1003.8	Networking
1003.9	FORTRAN Bindings

In addition to the formal committees, there are several informal committees made up of a cross section of the members from the formal committees who share common interests. Topics typical of those covered by the informal committees are lightweight processes and common language binding issues.

Historically, the term POSIX has referred to 1003.1, the C language binding. The term more properly identifies the collection of standards which will result from the group of committees.

4.2. Current State of Standardization

The committee has published its first work, an IEEE standard for the C binding to POSIX [POSIX88]. This standard has also been accepted and published as a Federal Information Processing Standard (FIPS).

The shell and utilities standard, representing the user interface to POSIX, is expected to be the next one published, having just recently undergone formal balloting.

4.3. Approach to Standardization

The 1003.1 standard is actually a C binding to POSIX. The System Interfaces committee chose to center its attention on the C binding because of the current widespread acceptance of these interfaces. To develop a language independent binding first would probably have delayed the committee so much that it may have been labeled as unproductive, resulting in reduced support for the standard under development, effectively killing the effort.

It is certainly undesirable to have a base standard in a family of standards which is highly slanted towards a particular language. Pressure has been applied from many

different directions to have a language independent functional specification developed which would form the basis for all language bindings, including the current C binding. Fortunately, this view is not held only by the Ada community. The International Standards Organization (ISO) will not adopt the C interface specification until a language independent definition is developed.

Under the auspices of the System Interfaces committee, the language independent effort has just recently started and is expected to be complete in 1991. A novel approach is being taken to the development of this standard in that the standard development is being done by a contractor under the supervision of the POSIX committees.

Unfortunately, this same approach of doing the C binding first and then following it later with a language independent specification has been adopted by some of the other committees as well, most notably the tools and utilities committee, the ones expected to produce the next published standard. The motivation for using this scheme is the same as for the 1003.1 standard: the C definition is already in existence and is thus easier and quicker to develop into a standard than an abstract definition. The problems this causes for the other language binding committees are that the other bindings are forced into an unnatural C orientation or the development of the binding is more costly because the essential nature of the C interface must first be extracted before a proper interface can be derived.

Ironically, one of the main forces driving this emphasis on quick development, and hence making these shortcuts, is the federal government. The National Institute of Standards and Technology (NIST) (US Department of Commerce) has established policies to accelerate standards development. The leverage they use is to establish a date by which a particular standard is to be established. If the standards committee is able to develop an acceptable standard by that date, then the committee's standard is adopted as a Federal Information Processing Standard (FIPS). If the committee is unready or unwilling to support the effort, then NIST will develop the standard itself. Concern about having an insufficient standard published forces the committees to take shortcuts which they normally would not adopt.

4.4. Ada Binding

Lack of a language independent specification is affecting development of the Ada binding. The Ada committee is the first non-C binding to be developed. Much of the effort in developing the binding is spent on determining what aspects

of the standard are necessary, as opposed to being artifacts necessary for the C definition.

The Ada community is resisting this approach very strongly. However, the potential users for a C interface far outnumber those resisting the approach. As a middle-ground approach, the members of the Ada committee are suggesting changes to the standard which will reduce the language dependence of the various standards, and therefore reduce the effort required to do the Ada bindings.

4.4.1. Ada Binding Philosophy

The Ada/POSIX binding philosophy is one which places the Ada aspects ahead of considerations of consistency between languages. To assist in definition of a philosophy, three styles of development were considered:

1. Try to make the Ada standard mirror the C binding as much as possible in terms of organization, architecture, and interface style.

2. Use the C binding as a guide to overall structure and concept, but do not consider it to be inviolate. Deviate wherever the application of Ada concepts would result in a standard which is more appropriate for Ada.

3. Define a complete Ada-oriented operating interface, adopting POSIX concepts where useful.

The Ada/GKS standard was developed primarily under the precepts of the first style. The resulting standard is one in which the various language bindings mirror one another very closely. A programmer trained in GKS in one language can be effective in the Ada binding very quickly. However, an Ada programmer who uses all of the features of the language finds that development style is different while creating GKS code from the style used elsewhere in the program. One example of the style difference is the use of return codes, rather than exceptions, to represent exceptional conditions. In the GKS style, code is written to analyze the return codes for each function call. In the Ada style, one writes a set of exceptions for a sequence of function calls, and writes no code to detect the conditions; they are detected automatically.

Development style (3) results in the ultimate Ada interface, but causes significant problems on three fronts. First, validating that the standard provides all functions necessary for a POSIX implementation becomes extremely difficult. A sufficiently distant abstraction can be impossible to verify. Second, the ability to demonstrate

that the standard is implementable, given some expectation of a POSIX functional description, is also made difficult. Without an understanding of implementability, a sufficient number of COTS implementations will not be made available. Finally, development time for the specification grows very quickly as the interface becomes more abstract.

Thus, choice (2) became the guide. With this model, it is expected that a high quality standard can be produced in a reasonable time and which exhibits a usage style which is natural to the Ada programmer.

It should be noted that the makeup of the committee dictates the style the standard will adopt. If the committee is mostly made up of individuals who are primarily motivated to get the most coverage for the subject of the standard (e.g. GKS, POSIX, or SQL), then a standard will tend to be motivated by consistency of binding to the subject and consistency across languages. If the committee is made up of experts in the target language, who are motivated by getting access to the subject, then the motivation will tend to be more supportive of coherent language style across subjects and will be less coherent between languages. It is clear that STARS, like most of the Ada community, is motivated towards consistent use of the language rather than ease of use in multi-lingual environments.

The extreme case of subject rather than language orientation was the original SQL standard, which was developed primarily by database experts. That standard implied to many readers that the recommended usage pattern was to embed the SQL code directly in the Ada source code, effectively extending the Ada language. The Ada community spoke out very strongly against the proposed standard, which was subsequently modified to support the module approach, a method which is much easier to deal with from an Ada perspective. The GKS binding is another example in which the development style was oriented towards the subject rather than the language. The resultant binding was valid according to the Ada precept of staying within the language, but still carried a very large FORTRAN orientation.

The POSIX standard is one of the first in which a very strong, but not radical, Ada approach is being taken. The committee consists primarily of members of the Ada community who desire a portable operating system interface for Ada, but have very little interest in maintaining consistency with other languages. In fact, the guiding precept of the committee is that it is reasonable to assume that the operating system underlying the Ada binding is written in Ada.

4.4.2. Development Plan and Progress

This first Ada/POSIX standard will encompass only the scope defined by the 1003.1 standard. Once that is complete, standardization of the other aspects of POSIX will be resolved. A step-wise approach is being taken for two reasons. First, the number of contributers who are able to make substantial contributions between meetings is small, typically around five volunteers. To obtain a standard of practical value, it is necessary to focus the committee resources on a reasonable goal. Second, and possibly more important, is that this standard is a test of the Ada-orientation described above. While it is fully expected that there is sufficient representation of the Ada community and sufficient oversight by other POSIX committees to ensure the binding will be acceptable, acceptance can only be completely determined by a public review of the standard. Rapid acceptance is expected due to support on the committee from three prominent Ada vendors (Alsys, VERDIX, and Meridian), two major defense contractors (Unisys and IBM), and many other prominent members of the Ada community.

Within this first step of producing a 1003.1 compatible standard, the development plan has two steps leading to public review. First, a binding for the "easy" parts of the standard is to be developed, followed by the "hard" parts. The "hard" parts are those in which there are basic conflicts between the Ada language and the operating system model which is highly C specific. Primary examples of the hard parts are interrupts and signals. Other parts which cause difficulty are the execution control (process) functions. Defining a relationship between the execution control functions and Ada execution semantics, including tasking, may not be possible. In such a case, the standard will leave this area to be implementation defined.

The binding for the easy parts is essentially complete and is undergoing final editing. The binding for the harder parts has begun with several proposals presented. It is expected that a public review can be held in 1990.

5. Conclusions

In the prior sections, the following conclusions were drawn:

- POSIX is important to STARS as a vehicle for building a portable SEE. It provides a virtual operating system layer which enables one to construct a SEE which is inexpensively ported.

- Commercial acceptance of POSIX, and Ada community support for the Ada binding to POSIX, combine to assure

COTS implementations will be available.

- From the perspective of a tool builder and application programmer in the SEE, POSIX is not visible -- it is completely masked by the interfaces provided by the project database and user interface management system.

- Because of the leverage one may gain by hosting the SEE on POSIX, it is very important that STARS continues to participate in and influence the POSIX standards effort, and in particular, the Ada binding to POSIX.

6. CAIS-A/POSIX Interface Analysis

The Unisys STARS implementation of CAIS-A which was completed during the first increment task (Q8) is hosted on Berkley BSD UNIX, exploiting some UNIX System V features which are distributed with SunOS. Not all of the interfaces used in this implementation are available in POSIX. The following is a list of the major feature groups, with specific interfaces identified, which are used in the implementation.

+ Shared Memory (for implementation of the server model)

  shmget

  shmat

  shmdt

  shmctl

+ Semaphores

  semget

  semopt

  semctl

+ Sockets (only used in the implementation of the distribution interfaces)

  bind

  accept

  connect

  listen

  select

  socket

Another feature which is not being used, but would be useful to use in the implementation is "Message Queues."

7. References

[CAIS-A88] Common Ada Programming Support Environment (APSE) Interface Set (CAIS) (Revision A), United States Department of Defense, May 1988. (Proposed DoD-STD-1838A).

[POSIX88] IEEE Standard Portable Operating System Interface for Computer Environments, The Institute of Electrical and Electronics Engineers, Inc., September 1988. (IEEE Std 1003.1-1988).

[RAC86] DoD Requirements and Design Criteria for the Common APSE Interface SET (CAIS), United States Department of Defense, October 1986. (Under NOSC Contract No. N66001-86-D-0156).