The Common Ada Programming Support Environment (APSE) Interface Set (CAIS)

PATRICIA A. OBERNDORF

Abstract—**This paper discusses the Common APSE Interface Set (CAIS) and its relationship to issues in the development of environments. The CAIS concepts and features are described, followed by a discussion of several ways in which the CAIS provides valuable capabilities for environment architectures and construction.**

Index Terms—**Interfaces, operating system interfaces.**

I. Introduction

THE Common Ada Programming Support Environment (APSE) Interface Set (CAIS) is a set of operating system level interfaces designed to facilitate the source-level transportability of tools between APSE's. For the purpose of the CAIS, transportability is defined as:

> "the ability of the tool to be installed on a different Kernel APSE (KAPSE); the tool must perform with the same functionality in both APSE's. Transportability is measured in the degree to which this installation can be accomplished without reprogramming" [2].

The interfaces are defined at the level at which tools customarily interact with operating systems. The CAIS provides interfaces to those traditional operating system services that affect tool transportability, such as file administration and manipulation as well as process control. The interfaces are specified as Ada procedures and functions and are presented as a group of Ada packages. The CAIS is intended to provide the transportability interfaces most often required by common software development tools. It is not intended to provide all interfaces that might ever be needed by all tools.

The DoD Ada Joint Program Office (AJPO) initiated the development of the CAIS in 1982. A group of international experts from government, industry and academia was tasked with defining the CAIS. This team, called the KAPSE Interface Team (KIT), defined the initial version of the CAIS over the course of about four years. The interface specification, designated DOD-STD-1838 [1], was approved as a military standard on October 9, 1986 and officially issued in 1987. In December 1985 a contract was awarded for definition of the first revision, DOD-STD-1838A, which is due for completion and standardization by the end of 1988. Prototyping work has been done by several different groups on a variety of host systems, and several Ada tools that use CAIS interfaces have been developed.

The original concept, motivation, and requirements for the CAIS were derived from STONEMAN [3]. STONEMAN defined the architecture for an Ada Programming Support Environment (APSE). This architecture is represented in Fig. 1. The innermost layer represents the underlying host machine and operating system. The next layer, the KAPSE, represents a layer of software which provides, in a host-independent manner, the host services required by tools. The tools of an APSE are divided into those minimally required to support Ada programming (e.g., the editor, compiler, linker), called the Minimal APSE (MAPSE), and the broader spectrum of tools needed to support software engineering in general (e.g., those supporting requirements, design, testing, document production), called the APSE. The CAIS can be thought of as the interface at the surface of the KAPSE layer, lying between the tools and the KAPSE software which realizes the CAIS services by translating them into the codes of the underlying host system.

The CAIS has been designed on these foundations and in keeping with more recent, evolving ideas about the requirements for integrated software engineering environments. Thus the CAIS provides many features that are important to the development of environments in general. The next section of this paper gives a technical overview of the CAIS, covering those features which are found in DOD-STD-1838 (Sections II-A–II-D) and then discussing (in Section II-E) the ways in which DOD-STD-1838A will build on these basic features. Section III describes how these features are of benefit in the construction of environments.

II. Technical Overview

The concepts underlying the CAIS are designed to provide a common, partial host system encapsulation that can be implemented on virtually any existing host operating

Manuscript received January 15, 1988. This work was performed under the sponsorship of the Ada Joint Program Office.
The author is with the Naval Ocean Systems Center, San Diego, CA 92152.
IEEE Log Number 8820969.

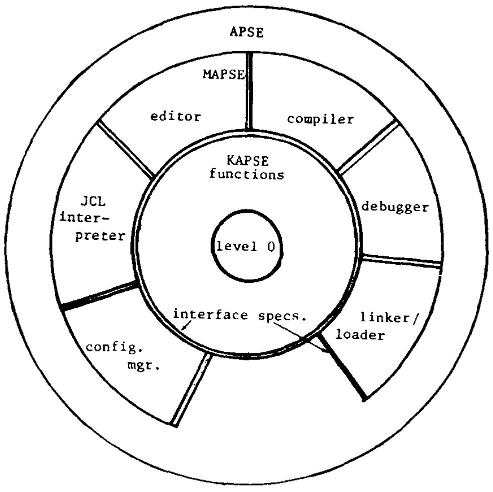

Fig. 1. Components of an Ada Programming Support Environment.

system or on a bare machine. This host encapsulation is a reflection of the common practice today for writing transportable tools.

Within this context, the CAIS provides a common, uniform, and simple system model, based on entity-relationship-attribute (ERA) concepts. Most current efforts to develop integrated environments utilize ERA models. Such models support the environment goal of making administrative information (i.e., information about the information) visible for tools in a common framework, thus facilitating transportability and integration of whole projects. The ERA approach allows the user to define entities, the interrelations between them, and the properties (attributes) known about the entities and the interrelations. In this way information which today is typically encoded in naming conventions or registered only in someone's memory is made explicit in the database. The common model avoids the dangers of multiple tools maintaining duplicate (and possibly inconsistent) data while providing a framework for integrating tools. The uniformity and simplicity are intended to make it natural and comfortable for tool writers to develop CAIS-hosted tools.

The ERA model provides a basis for tools to share and understand common data. It makes it possible for tools to operate effectively on all shared administrative information in an implementation across homogeneous and even heterogeneous host systems. This explicit sharing of information establishes another aspect of the framework in which additional integration issues can be addressed.

The CAIS is designed to be open-ended and capable of evolving over time as the technologies of environments and host systems evolve. The interfaces are also designed to be usable by any tool, regardless of methodological considerations. They allow access to underlying host operating system capabilities where needed while imposing almost no restrictions on tool writer methodology. This open-endedness is essential to providing the basis for the realization of the goals for modern (current and future) environments.

There are three basic groups of interfaces which provide the most important features of the CAIS. These support the node model, process management, and input and output.

A. The Node Model

At the heart of the CAIS is an ERA-based concept called the node model. It provides for the administration of the entities which are commonly dealt with during software development projects, such as files, processes and devices. In the node model, each such entity is represented by a *node*. Interrelations between nodes are represented by *relationships*. Both nodes and relationships have attributes describing their properties. In addition, nodes may be used to group other nodes in the node structure, not directly representing a file, process, or device themselves. In this way they can be used like directories in conventional operating system file systems. Within this framework it is possible to administer the people, items, and activities which make up software development projects in a very natural way. An example of a CAIS database using the node model is shown in Fig. 2.

Relationships can be either *primary*, providing a hierarchical back-bone to the node structure, or *secondary*, providing a network orientation. Every node is the target of exactly one primary relationship, but secondary relationships have no such restriction. There is a conceptual root node to the overall hierarchy, thus providing (via primary relationships) at least one unique path to every node. Nodes are identified by the concatenation of the names associated with the relationships along a path from some starting node ending at the desired node. Since always having to use full pathnames could result in significant performance penalties, two mechanisms for pathname abbreviations are provided.

Interfaces are provided to perform the functions which are expected. Thus tools can create, copy, delete, and rename nodes and copy and delete subtrees (based on the hierarchy provided by the primary relationships). Both predefined and user-defined relationships and attributes can also be created and manipulated. Interfaces are provided for iterating over the relationships or attributes of a given node.

The CAIS definition provides several predefined attributes which provide special information about nodes. Chief among these are the timestamp attributes which record when a node was created and when the relationships, attributes, and contents were last written. Such attributes are key to most configuration management schemes.

Support for both discretionary and mandatory access control, as defined in [4], is provided in the CAIS. Access to nodes is controlled at the level of the whole node and, for discretionary access control, at the granularity of contents, attributes, or relationships. Access control is represented using the node model and so is completely integrated into the CAIS structure. Processes can change roles

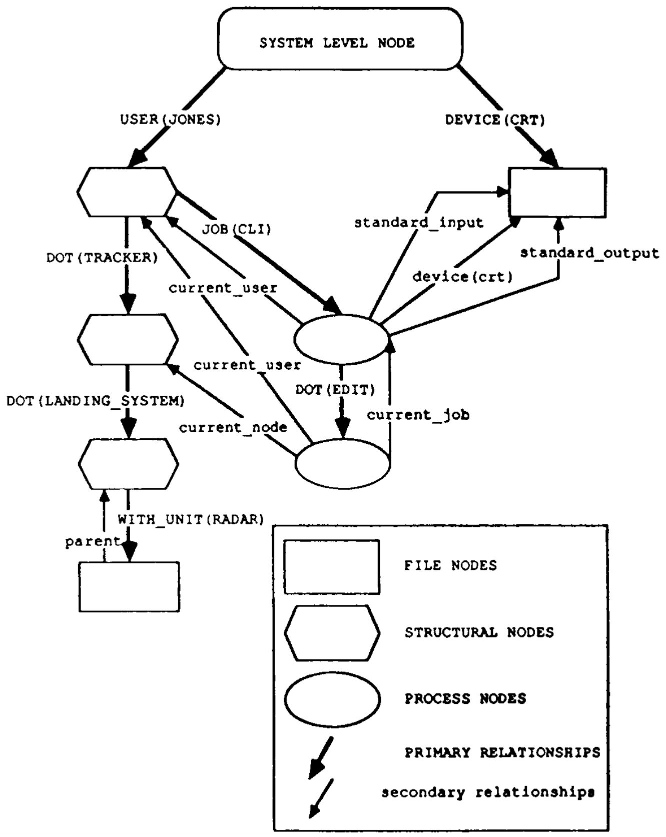

Fig. 2. CAIS Node Model Example.

and set access control information using interfaces which modify relationships and attributes. Access synchronization is provided at the same granularity as discretionary access control for establishing exclusive access, i.e., locking against simultaneous access by two different processes. The ability to lock at this level of granularity has distinct performance advantages.

B. Process Management

The CAIS interfaces support a process model. Processes are also represented as nodes in the node model, so process naming is consistent with the naming of all entities in the information base. Most of the information in Section II-A applies equally to the process nodes. A CAIS process represents the execution of an Ada program. The CAIS process model provides the ability to *spawn* (i.e., continue in parallel with) or *invoke* (i.e., wait for) a child process. Other process control operations such as suspend, resume, await and abort are also provided. Both dependent processes (i.e., a process which cannot continue to exist if its creator no longer exists) and independent processes (i.e., one that can continue after its creator has ceased to exist) can be created. Several predefined attributes peculiar to process nodes are supported, giving such information as the times at which the process started and finished, the machine-time consumed and the process size (i.e., the amount of memory currently in use by the process).

It is important that the CAIS provide a process model in addition to Ada's tasking model because there are some important differences between task interaction within a single program and process interaction between independent programs. The most critical difference is that task interactions require that the tasks be compiled together within a single program. An environment cannot afford to be limited to such a paradigm. It would require that portions of the environment be recompiled and the entire environment be relinked every time a new, concurrently executing tool is added, which would be entirely unacceptable. In today's environments, it cannot even be guaranteed that separate tools will be compiled with the same compiler, let alone as part of the same monolithic program. Instead, an environment must provide a means of process interaction which depends only on what independent programs can be reasonably expected to know about one another.

C. CAIS Input and Output

The CAIS provides several basic forms of input and output. First, it supports forms of DIRECT_IO, SEQUENTIAL_IO, and TEXT_IO which are very nearly identical to what is provided in Chapter 14 of the Ada

Reference Manual [5] and which can be used to implement Ada input and output on top of the CAIS. The main differences are in the additional semantics required in order to maintain consistency in the underlying CAIS information base.

Secondly, the CAIS supports three basic types of terminals: scroll (e.g., TTY's), page (e.g., VT100's) and form (e.g., IBM 327x). These support the main kinds of user interactions which are found in many of today's environments, as appropriate to the features of the respective terminal types. All three of these terminal packages are based on ANSI standards.

Thirdly, the CAIS supports the ability to control and make use of magnetic tape drives, including the abilities to mount and load tapes and to query various status indicators. This, too, is based on ANSI standards.

Finally, the CAIS supports the ability to export files to the underlying host file system and to import files from the underlying host file system to the information base.

All files and devices in the CAIS are represented as file nodes (i.e., nodes whose contents are Ada external files). While device nodes have some special characteristics (e.g., arbitrary users cannot create them), they are designed to fulfill all of the basic needs of tools for input and output.

D. Other Features

In addition to the three basic groups of interfaces discussed above, the CAIS supports the processing of *lists*, which are used in the CAIS to represent values of attributes. The CAIS also supports the observance of certain *pragmatic* limitations which are important for the achievement of transportability. If the CAIS did not specify some practical limitations, then it might become very difficult to achieve transportability between some combinations of different implementations (e.g., a tool implemented on a host which supported very liberal limits would not be able to perform correctly on a host with very restricted ones without the ability to query these values and adjust its processing accordingly).

E. Future Features

DOD-STD-1838 is an initial version of the CAIS, and, as was indicated in the introduction, work is already underway on Revision A (DOD-STD-1838A). In parallel with the development of DOD-STD-1838, a set of requirements was developed which embodies the most widely accepted definition to-date of what is required for a transportability interface set which provides an appropriate framework for future integrated environments. The features listed below are the most significant ones which do not appear in DOD-STD-1838 but are part of the requirements set for DOD-STD-1838A:

- user-defined typing of the node model, including explicit representation of the typing information in the node model and inheritance of types
- triggering (i.e., the ability of defined events in the database to trigger further actions)
- definition of a common external form designed to facilitate the movement of node model databases from one environment to another
- transactions
- explicit support for distribution
- more devices for input and output, including more sophisticated terminal capabilities.

These features will serve to increase the sophistication of the support available from the CAIS. By and large, they are features which are either currently in use in today's environments or which have been identified as being important to architectural concepts for the environments of the future. In the work now underway to define DOD-STD-1838A, all effort will be made to add these features in a way that is upwardly compatible with DOD-STD-1838.

III. Relationships to the Development of Integrated Environments

The CAIS is designed for use in the construction of integrated environments. Although the driving motivation has been Ada tool transportability, there are many aspects of the CAIS design which can be of benefit in the design and construction of environments in general. Some of these are related to architectural issues and others to construction approaches. These are discussed in Sections III-A and III-B. Section III-C discusses some more intangible benefits of using the CAIS in building environments and compares some of these aspects to other similar interface sets. Section III-D discusses some of the more concrete benefits.

A. Conceptual Relationship to Environment Architectures

One significant area of architectural agreement among most environment projects regards the need for a central database capable of supporting large projects in the capture and retention of all of the information developed and required during system development and maintenance. The CAIS provides such a central database capability. The ERA model on which CAIS is based is probably the most widely accepted database model for use in environments today. With the node model, the CAIS supports in a natural way the kinds of recording, interactions, and management techniques required for large projects to be successful today. Thus the CAIS can play a central role in the architecture for almost any modern environment.

As a part of the node model, the CAIS incorporates support for discretionary and mandatory access control. While many software support systems provide some basic form of discretionary security, few environments today have adequately addressed the needs for mandatory security or more demanding discretionary security. But even projects outside of the DoD are beginning to recognize requirements for greater security than before, and mandatory security is something that cannot be left to be added on "later." It must be architected into the system from the beginning. The CAIS provides an appropriate basis for environments that have this requirement. At the same time, the specification of security features in the CAIS is such that there is no penalty to the performance of the environment if support for mandatory access control is not required.

Another essential feature which the CAIS can offer to the architectures of today's integrated environments is a common substrate and framework for discourse. With the CAIS a common basis is provided and effort can be concentrated on the very difficult issues in tool integration which builds on this basis. In particular, the CAIS node model provides a paradigm for tool interaction upon which such things as intertool interfaces and conventions for information sharing can be based.

Flexibility is another CAIS feature which is quite important architecturally. While providing a common basis for integrated environments, the CAIS remains quite general. It does not impose any preconceived ideas regarding methodological approaches (e.g., use of structured analysis/design), management disciplines, or tool-writing techniques. It provides the primitives necessary for environments to achieve all that they require without dictating any of the policies. This is important for the production of effective, usable environments because flexibility is a key feature for supporting different methodologies and project management approaches. No one environment will be appropriate for all time for all projects. Environment architectures must accommodate this variation in needs through flexibility, leaving the imposition of specific policies to the ways in which tools and projects utilize the primitives.

Distribution of an environment over a variety of hardware resources is a fact of life for today's systems. The CAIS has been designed so as not to preclude distributed implementations. The CAIS design in [1] does not support the explicit control by tools over the distribution (although future versions will; see Section II-E), but it also does not require that tools be aware of whether or not the underlying hardware is made up of one processor or many or where a particular device is located. This increases the flexibility of the architectural basis provided by the CAIS.

B. Relationship to Construction of Environments

Because most environments of today are being constructed on top of conventional operating systems and CAIS has been designed with this level of interaction in mind, it can be said to provide a (partial) virtual operating system for the implementation of tools. Thus it fits naturally into today's usual approach to constructing environments. Experience so far [6] has shown that tools written in Ada can fairly readily be converted over to the use of the CAIS from use of the host operating system for which they were originally designed.

The CAIS objective of transportability is an important feature. One of the key issues in providing quality environments to a wide variety of projects is how to cost-effectively obtain the broad spectrum of tools which is required for lifecycle support. There is far too much work to be done for any one organization (even the U.S. DoD) to do the whole job of generating all those tools alone. Only through sharing and common availability of a variety of tools can individual organizations ever hope to achieve the sorts of lifecycle-support environments that are required. The CAIS can provide the common basis which makes wide availability of tools, and therefore tool sharing, practical. In addition, through extensive use by a variety of projects, the maturity and therefore the quality of our tools will increase.

The CAIS has also been designed to be open-ended and extensible. That is, it does not claim to provide all interfaces that will ever be needed by any tool. Too many specialized interfaces simply cannot be provided in transportable ways, and there are often so many desirable choices (e.g., which graphics standard to provide) that it would be too limiting to provide one and not support others. By being extensible itself, the CAIS facilitates extensibility of environments, a critical feature at a time when complete, fully integrated environments are still a goal and rarely a reality.

The open-endedness of the CAIS also implies that arrangements can be made for non-Ada tools to properly interact with a CAIS-based environment. While this is best facilitated by an Ada compilation system that supports the use of programs written in other languages, there are other approaches. It should be noted that, although the CAIS interfaces are specified in a style which takes advantage of and is consistent with Ada, it is not required that they be implemented in Ada.

C. Benefits for Building Environments

There are several benefits which the CAIS provides when used as a basis for building environments today.

It is possible to implement the CAIS without requiring dedication of the host machine(s); in other words, a CAIS implementation can coexist with another previously existing operating system. This means that the wide availability needed is possible with CAIS, and it does not require that every project invest in new suites of equipment.

The achievement of tool transportability through the CAIS has additional benefits beyond making a lot of tools available. Normally tool writers devise their own host-encapsulations in order to achieve transportability of their tools. While this achieves transportability of each tool by itself, it does not provide a common basis for a tool suite and thus for future integration with other tools (unless the same tool writer is developing all tools in the suite and so writes all the tools to the same host encapsulation). There is also considerable duplication of effort across all such tools and tool writers. Use of the CAIS host-encapsulation instead, even when a CAIS implementation is not available, would result not only in a reduction of the tool writer's design effort, but also in a potential for future rehosting to the CAIS without any potential disadvantages from retrofitting.

People portability is another benefit of the CAIS transportability feature and the commonality which results. Tool writers can avoid perpetually learning new systems and better concentrate their energies on writing superior tools for a common basis. In addition, as people move from one project to another, they can move their favorite tools with them, thus saving the time required to relearn a whole new environment. They can become productive immediately, and productivity is one of the most important objectives in building environments.

There are a few other interface sets which claim some of the same benefits as described here for the CAIS. Most notable among these is the Portable Common Tool Environment (PCTE) [7] defined by a consortium under a contract with the Commission of European Communities (CEC) European Strategic Programme for Research and Development in Information Technology (ESPRIT). PCTE is a UNIX®-oriented common interface which fulfills many of the same goals as the CAIS. There are versions of the interface set in both C and Ada. Another alternative mentioned frequently is UNIX itself. Both of these share most of the benefits discussed above. But, when compared to these and others, the CAIS has some benefits which set it apart.

The CAIS has been carefully designed to be independent of any particular operating system. This provides a degree of vendor independence not possible with the others. More importantly, it provides a more modern and adaptable basis for environments than UNIX or any UNIX-based approaches can. Without the central database provisions of CAIS and PCTE, UNIX cannot provide the support required for modern integrated software engineering environments. An ordinary file system, even a hierarchical one such as that found in UNIX, cannot provide the robustness of the ERA model or the explicitness of information representation which the ERA model supports.

A major difference between the CAIS and both PCTE and UNIX is in its support of security requirements. The current version of PCTE supports no such requirements, and UNIX has achieved a level of support for such features only through the definition of restricted kernels.

The CAIS is also a controlled Department of Defense (DOD) standard. This means that it is stable and subject to well-known procedures which utilize public inputs to evolve the standard as technologies advance. It has also been specified with sufficient rigor to make validation of implementations possible, and a validation suite is under development. Standardization plus validation means that the commonality that is so attractive can be achieved, maintained, and enforced. In contrast, PCTE has not been standardized, although there is only one set of definers and a control board is now in place, under the auspices of the CEC. However, public input to this control process is limited. UNIX has the opposite problem: there are so many candidates for "standard" UNIX that there is no single standard. There is a new IEEE trial-use standard (known as POSIX), but there is also AT&T's System V and the de facto "standard" of Berkeley 4.3, just to mention a few of the contenders. Because PCTE is UNIX-based, it potentially falls victim to this same confusion.

The CAIS has limitations as well. One is the fact that the only language binding of CAIS is Ada-oriented, which is understandable since the sponsor of the work has been the Ada Joint Program Office. For the purposes of the DoD, this is quite adequate, as Ada has been directed to be the common language for defense systems. For the purposes of the general community, this is an acceptable limitation, as Ada provides a very robust foundation for software engineering activities in general and has the language features necessary for interfacing with other languages.

Another aspect which might be called a limitation of the CAIS is that compromises have been made on some features in the interest of creating an interface set which can be implemented effectively on a wide range of existing operating systems. This is in contrast, for example, with PCTE, where the adoption of UNIX means that there are no considerations of generality or compromises to be made. However, one pays for this lack of generality every time one attempts to implement a feature that is unique to UNIX on some other operating system. If the objective is to make an environment widely available, the occasional compromises are worth the generality.

The CAIS does not attempt to define either intertool or user interfaces. These can be considered to be outside the scope of CAIS, but they are also important to the production of integrated environments. Unfortunately, today there is far too little agreement on any of these to be able to establish viable standards. The identifiable user interface standards are evolving far too quickly, and too little is known about which intertool interfaces should be standardized or how these should be defined. The CAIS does not attempt such definitions, but rather provides a flexible, extensible backdrop for experimentation in these newer areas.

D. Available Resources for Building Environments with CAIS

Several CAIS prototype implementations exist today. Most implement various-sized subsets of the standard, and many are hosted on VAX® systems, some running UNIX and others VMS. Some other non-VAX implementations also make use of UNIX or UNIX-look-alikes. A variety of tools has been hosted on these prototype implementations, including a compilation suite. Some tools have been rehosted from other operating systems, while others have been written directly for the CAIS. The DoD is currently sponsoring two full near-production-quality implementations of DOD-STD-1838 which will be completed during 1988. One will be hosted on VAX/VMS and the other on IBM 370/MVS.

®UNIX is a registered trademark of AT&T Bell Laboratories.

®VAX is a registered trademark of Digital Equipment Corporation.

IV. Conclusion

The motivations and basic features of the CAIS have been discussed, along with its relationships to architectures and the construction of environments. In the last three years much experience has been gained with CAIS implementations and what is necessary to make sound use of the CAIS features in the development of integrated software engineering environments. It is anticipated that in the near future there will be substantial evidence that the CAIS makes significant contributions to the development and evolution of software engineering environments.

V. References

[1] U.S. Dep. Defense, *Military Standard Common Ada Programming Support Environment (APSE) Interface Set (CAIS)*, DOD-STD-1838, Oct. 9, 1986.

[2] P. A. Oberndorf, *KAPSE Interface Team: Public Report*, vol. 1, Apr. 1, 1982, p. C1.

[3] J. N. Buxton, *Requirements for Ada Programming Support Environments, "STONEMAN"*, U.S. Dep. Defense, Feb. 1980.

[4] U. S. Dep. Defense Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, Aug. 15, 1983.

[5] U.S. Dep. Defense, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, Jan. 1983.

[6] M. R. Gardner, R. L. Hutchison, and T. P. Reagan, "A portability study based on rehosting WIS Ada tools to several environments," WP-8600396, MITRE Corp., Sept. 30, 1986.

[7] European Strategic Programme for Research and Development in Information Technology, "*PCTE, A basis for a portable common tool environment*," in *Functional Specifications*, 4th ed., 1986.

Patricia A. Oberndorf received the B.S. degree in mathematics and computer science from Oregon State University, Corvallis, in 1971, and the M.S. degree in computer science from the University of California at San Diego in 1974.

In 1974 she started work at what is now the Naval Ocean Systems Center (NOSC) in San Diego. Her first involvement with software engineering environments was the result of the Systems Design Laboratory (SDL) project, on which her assignments included design, implementation, introduction of tools such as URL/URA, and consultation with users. In 1978 she traveled to Europe at the request of the London branch of the Office of Naval Research to survey relevant research and development work in England, Germany, and France. She has been involved in the Pebbleman/Stoneman process and has represented the Navy and the DoD in many environment-related meetings and conferences. She was the Chief Engineer and shared management and planning responsibilities for the Software Engineering Automation for Tactical Embedded Computer Systems (SEATECS) project before accepting responsibility for the definition of the CAIS in 1982. Her present assignments also include continual investigation and assessment of research and development work relating to software and system engineering.