N89 - 16319

540-61
16004
14P

W... 533

TRANSPORTABILITY, DISTRIBUTABILITY, AND REHOSTING EXPERIENCE WITH A KERNEL OPERATING SYSTEM INTERFACE SET

F. C. Blumberg, A. Reedy, and E. Yodis
Planning Research Corporation
1500 Planning Research Drive
McLean, Virginia 22102

INTRODUCTION

For the past two years, PRC has been transporting and installing a software engineering environment framework, the Automated Product Control Environment (APCE), at a number of PRC and government sites on a variety of different hardware. The APCE was designed using a layered architecture which is based on a standardized set of interfaces to host system services. This interface set, called the APCE Interface Set (AIS), was designed to support many of the same goals as the Common Ada™[1] Programming Support Environment (APSE) Interface Set (CAIS): transportability of programs; interoperability of data; and distributability of the environment processes and data. However, the evolution of the AIS has been quite different than that of the CAIS. The AIS was designed to support a specific set of lifecycle functions and to provide maximum performance on a wide variety of operating systems.

The APCE was developed to provide support for the full software lifecycle. Specific requirements of the APCE design included: automation of labor intensive administrative and logistical tasks; freedom for project team members to use existing tools; maximum transportability for APCE programs, interoperability of APCE database data, and distributability of both processes and data; and maximum performance on a wide variety of operating systems. The functions supported by the APCE include: configuration management for lifecycle products; traceability; change and release control; project control and reporting; management for all levels of testing including integration and system testing; and support for standards enforcement. The AIS design is critical in supplying transportability, interoperability, and distributability. The AIS design is also critical in providing the basis for APCE performance.

This paper gives a brief description of the APCE and AIS, a comparison of the AIS and CAIS both in terms of functionality and of philosophy and approach, and a presentation of PRC's experience in rehosting the AIS and transporting APCE programs and project data. Conclusions are drawn from this experience with respect to both the CAIS efforts and the Space Station plans.

[1]Ada™ is a registered trademark of the U.S. Government Ada Joint Program Office.

ENVIRONMENT FUNCTIONS

The APCE has been designed based on a separation of concerns between the functionality of the environment framework or architecture and the functionality of tools. The environment provides control, coordination, and enforcement of standards and policy and acts as repository for information (including software lifecycle products). The tools assist the project members in the actual creation or modification of the products (software and associated documentation and lifecycle products).

The APCE supports a software lifecycle process paradigm. The software lifecycle is viewed as a series of development or maintenance projects. Project members fall into three board categories: managers, developers, and testers. Developers include all project members who create or modify lifecycle products: requirements analysts, designers, coders, etc. Testers include the traditional categories of configuration management and quality assurance personnel and personnel involved in product reviews and audits. Projects have phases which can be defined in terms of the products developed during each phase. The APCE requires a testing process for the products of each phase. The paradigm is illustrated in Figure 1 which uses Mil-STD-2167 phases and products as an example. The APCE is configurable for different phases and products as well as for different methods of integrating products (software or documents) from components.

The functions provided by the APCE framework include:

- o configuration management of software, documentation, and test procedures;
- o automated status reporting and tracking of product components, work packages, and changes;
- o maintenance of traceability from requirements through development to code;
- o automated test bed generation and support for testing from unit testing through system testing;
- o maintenance of project database;
- o automated integration and release control for products;
- o enforcement of selected standards and procedures through testing;
- o project specific environment configuration.

The user interface consists of a set of menus for the major subsystems. The functions provided by the five major subsystems are summarized below.

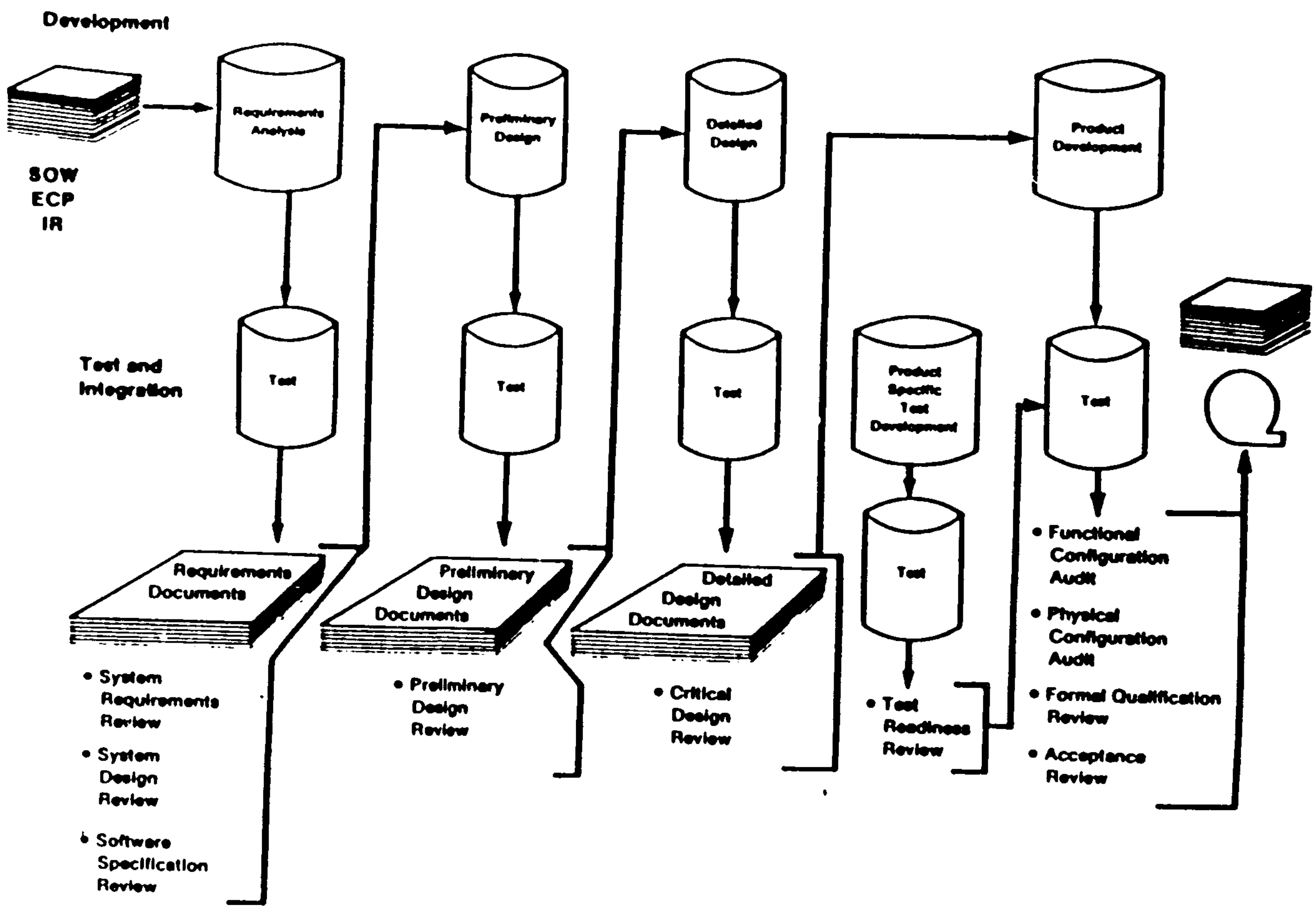

FIGURE 1: APCE DoD DOCUMENTATION AND REVIEW SEQUENCE

<u>Generation Subsystem</u>: The generation subsystem allows selected privileged users to configure the APCE to the specific project in terms of user groups and organization, work packages and schedule, project phases, products, and product integration structure. The APCE can be reconfigured to reflect changes to the project structure and organization as needed. The generation subsystem uses this information to organize the project database.

<u>Development Subsystem</u>: The development subsystem allows developers to select the data or products associated with their task and to return their finished products back into the database when they are ready for testing. The developers can use the software tools available on their host system or workstation to work on the products. The current version of the APCE does not directly control the use of these tools.

<u>Test Subsystem</u>: The test subsystem supports the testers in the building, execution, and reporting of the product tests. The test subsystem allows the testers to create test procedures, which are then managed by the APCE. The APCE will build test beds and integrate product components for the testers, who will then execute tests. The testing process provides the methods for enforcement of standards and policies. The testers report the test results through the test subsystem. Testers are also responsible for system release in the APCE paradigm and the test subsystem performs this function.

<u>Change Control Subsystem</u>: The change control subsystem allows managers to enter change requests into the system and to define stop dates for release support.

<u>Report Subsystem</u>: The report subsystem allows managers and other APCE users to get reports on the current status of changes, test procedures, and releases. It also gives reports on project status by task or by product component. Additional reports provides impact analysis for proposed changes and other traceability information.

<u>ENVIRONMENT GOALS</u>

The goals of the APCE design are:

- o to provide management and control for the full software lifecycle process;
- o to automate the labor intensive administrative and logistical overhead functions;
- o to allow full use of existing hardware, operating systems, file management/DBMS, and communication resources.

The last goal implies a series of subgoals. An environment should be distributable across heterogeneous operating system configurations, heterogeneous file management/DBMS facilities and use the available communications facilities as well as heterogeneous hardware configurations.

The control framework must be easily transportable to new hardware and host systems at reasonable cost. The environment database, including the lifecycle products and their relationships and attributes, must be easily moved between environment instances. There must be no performance penalties for using the environment. It must cooperate at some level with existing operating systems to take advantage of their security and performance features. Finally, the environment must allow the use of existing software tools and allow flexibility for retooling as necessary.

ROLE OF THE APCE INTERFACE SET

The basic architecture of the APCE is best described as "Stoneman inspired but data coupled". The system is layered as illustrated by Figure 2. The host system (s) provide basic services such as operating system services, file management system/access mechanism or database management system, access controls, and communications mechanisms as needed for the configuration. The communications facilities are needed if distribution, workstations, or remote test beds are desired. The software engineering environment instance based on the APCE is layered on top of these services. The instance provides users with project specific tools and procedures which will usually exercise the host services directly and the APCE major subsystems which exercise the host services through the APCE Interface Set (AIS).

Since the APCE major subsystems use AIS calls, the APCE is transported to a new hardware/OS/DBMS configuration by rehosting the AIS. Thus, the AIS provides the Kernel interface described by Stoneman and supports the goal of distribution. Since all database accesses must be made through the AIS, the AIS also supports the interoperability of project data.

IMPLEMENTATION PHILOSOPHY

The AIS design reflects the implementation philosophy of the APCE as a whole. The architecture of the APCE is data coupled. That is, the APCE subsystems do not interface directly with each other; rather, they interface via the AIS to the project database. The APCE adopts an open system approach towards the use of third party tools. The APCE controls lifecycle products which are entered into the database through user interaction with APCE subsystems. Thus, there are no constraints on the tools used to develop the products. For maximum performance, the AIS is designed to function in conjunction with a modern operating system rather than on a bare machine. Tools do not have to be rehosted to the AIS in order to be used.

The AIS was developed by defining a set of transportability rules that provide the maximum independence for applications (tools, programs, etc.) from the run time environment. **For maximum transportability**, it was determined that **the application must have a logical view** of the operating services, the database services, communications services and the data it uses. The industry is evolving toward this conclusion, however, only a step at a time.

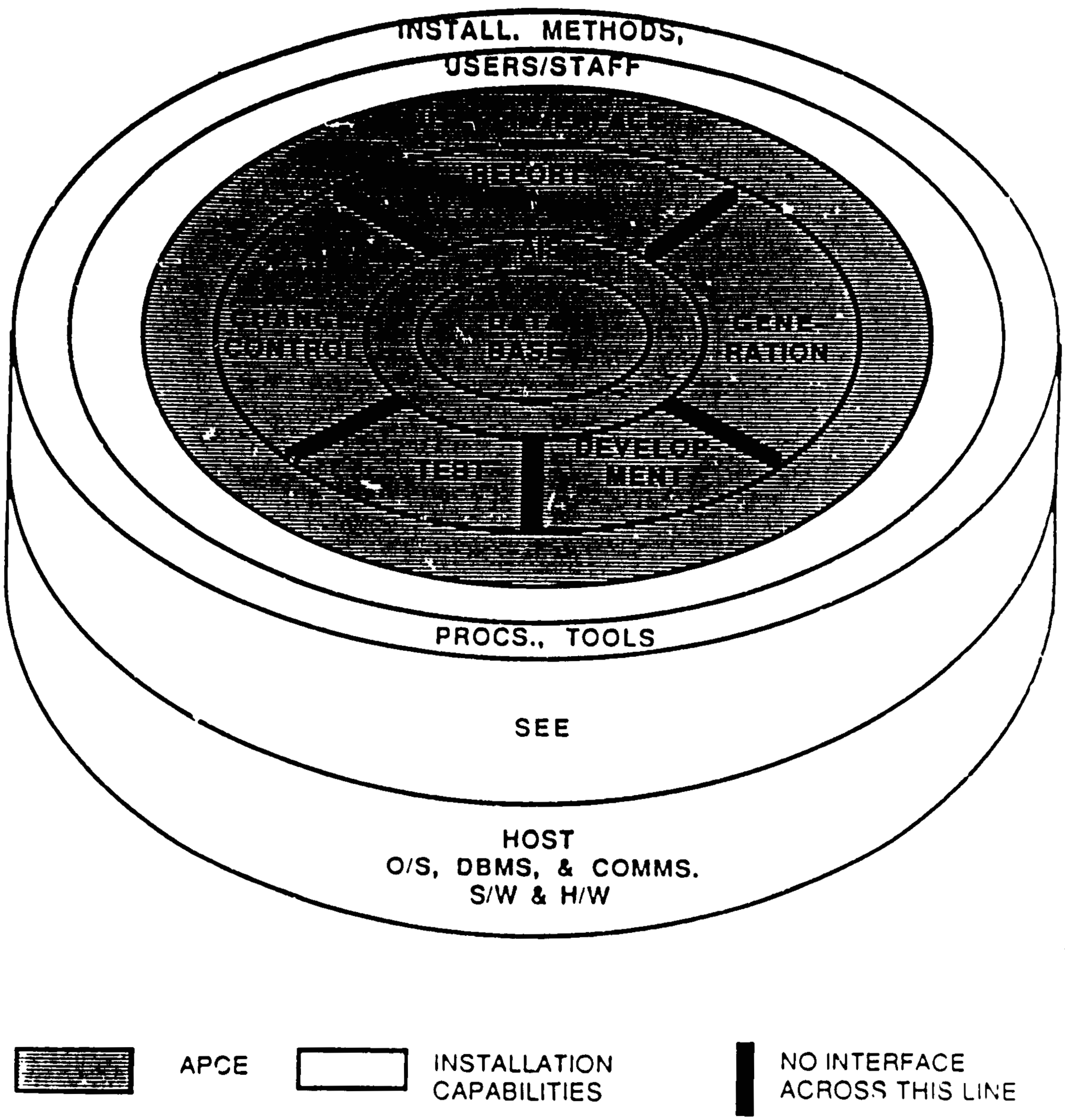

FIGURE 2: APCE STATIC VIEW

As an example, currently UNIX™[2] is considered transportable and it does provide hardware independence. However, it does not provide application independence any more than any other operating system. Accepting a operating system as the basis for transportability provides the application a highly constrained set of system services, database services, and communication services which may adversely affect the applications performance. Therefore a set of logical service interfaces was implemented that can be mapped to any operating system, file management/database management system and communication protocols.

This AIS implementation has been proven transportable over a wide range of operating systems, file management/database management systems, and hardware. The AIS design approach assumes that the host system has been developed by the vendor to take full advantage of the hardware features of the computer. The host system should provide performance achievable only through intimate study of the hardware system. The AIS takes advantage of the host system performance and does not try to duplicate it. The performance the AIS should be the same as that of the services supplied by the host system

The AIS assumes that the following features are supplied by the host system:

- o file management system/access mechanisms or database management system;
- o access controls;
- o command processor with command script feature;
- o communications mechanism (e.g. VAX™[3] DECnet) between host(s)/ workstations(s)/targets(s) if distribution or remote workstations or remote test beds are desired.

CAIS/AIS COMPARISON

The CAIS had no impact on the APCE development, however both the CAIS and the AIS had similar goals. The intent of both interfaces sets was to achieve transportability of tools between environments and to achieve interoperability of data between environments. The CAIS was in response to a need in the DoD for cost reduction and commonality of tools for software development. The same requirement fostered the AIS developed within PRC. PRC has many software development contracts running concurrently, and each contract has different required hardware, tools, and methods. Therefore, PRC requires an environment that is adaptable, transportable and allows interoperability of data and excellent performance on any host system.

The AIS strategy is based on a layering of system services rather than on a specific system service interface model (such as the node model of the CAIS).

[2]UNIX™ is a registered trader ark of Bell Laboratories.

[3]VAX™ is a registered trademark of Digital Equipment Corporation.

The APCE software is based on an interface into which the host system services that satisfy the interface specifications are mapped. The AIS design is based on the expected availability of certain host system services. If a service is not directly available, then extra layers of software which provide the needed enhancement are created below the interface layer to satisfy the requirement.

Both the CAIS and the AIS attacked the problem at the interface layer between operating system services and the application programs. See Figure 3, AIS/CAIS Comparison, for AIS/CAIS comparison. As the diagram illustrates, the AIS provides services at a slightly higher level of abstractness than the CAIS. In addition, the AIS already has additional interfaces operational (DBMS, Communications) that the CAIS has not implemented as can be seen in Figure 4, CAIS/AIS Major Functions. The CAIS also requires a significantly greater number of functions primary because of the node management requirement. The AIS terminal I/O implementation currently only handles form management functions, and therefore does not provide as rich a set of features as the CAIS terminal I/O provides.

The primary difference between the AIS and the CAIS is the concept of the node model. The node model provides a method of organizing files, directories, devices, queues, and processes into a form that can be manipulated by any APSE tool on any host that implements the CAIS. The node model is similar to the implicit node model within the UNIX™ operating system with some extensions. The AIS embraces the concept that applications (programs, tools) require only a logical view of the services. Therefore, the interface functions should be mapped into the existing system services providing these capabilities.

The AIS provides only the logical view of the system services to the application which accomplishes two goals, total application independence and improved performance. Figure 5, CAIS/AIS Implementation Differences, illustrates each implementation.

Application independence is attained because dependence on structural or physical implementation of each service has been removed from the applications domain. This has not been attained in the CAIS because each application has knowledge of the node model and therefore any change to the node model will require a change to all applications dependent upon that structural knowledge.

The direct mapping of AIS services to system services enables an AIS implementation to operate as efficiently its host system. The CAIS, however, superimposed a control structure (the node model) on top of existing services that may limit performance on a given CAIS implementation.

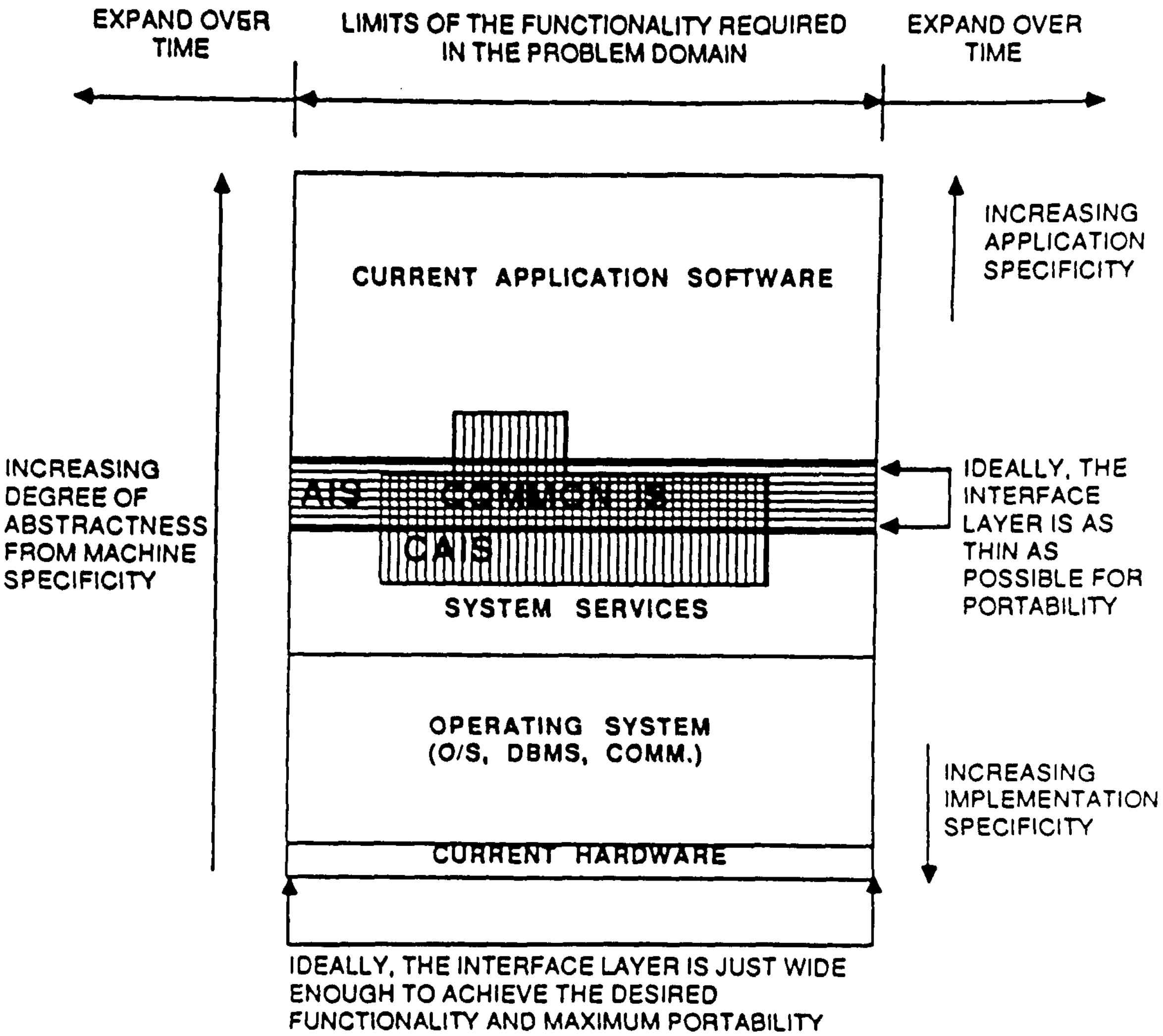

FIGURE 3: AIS/CAIS COMPARISON

	CAIS MAJOR FUNCTIONS	AIS MAJOR FUNCTIONS
SYSTEM MANAGEMENT (Node Management)	73	NONE. These functions are accomplished differently by each O/S, DBMS, and Communication suite.
PROCESS		
Host	32	15
Distributed	Not Implemented	37
INPUT/OUTPUT		
File Access/DBMS	38	54
Terminal	106	14
Tape	20	Host Operating System Provided
Distributed	Not Implemented	29
UTILITIES	86	26
TOTAL	355	175

FIGURE 4: CAIS/AIS MAJOR FUNCTIONS

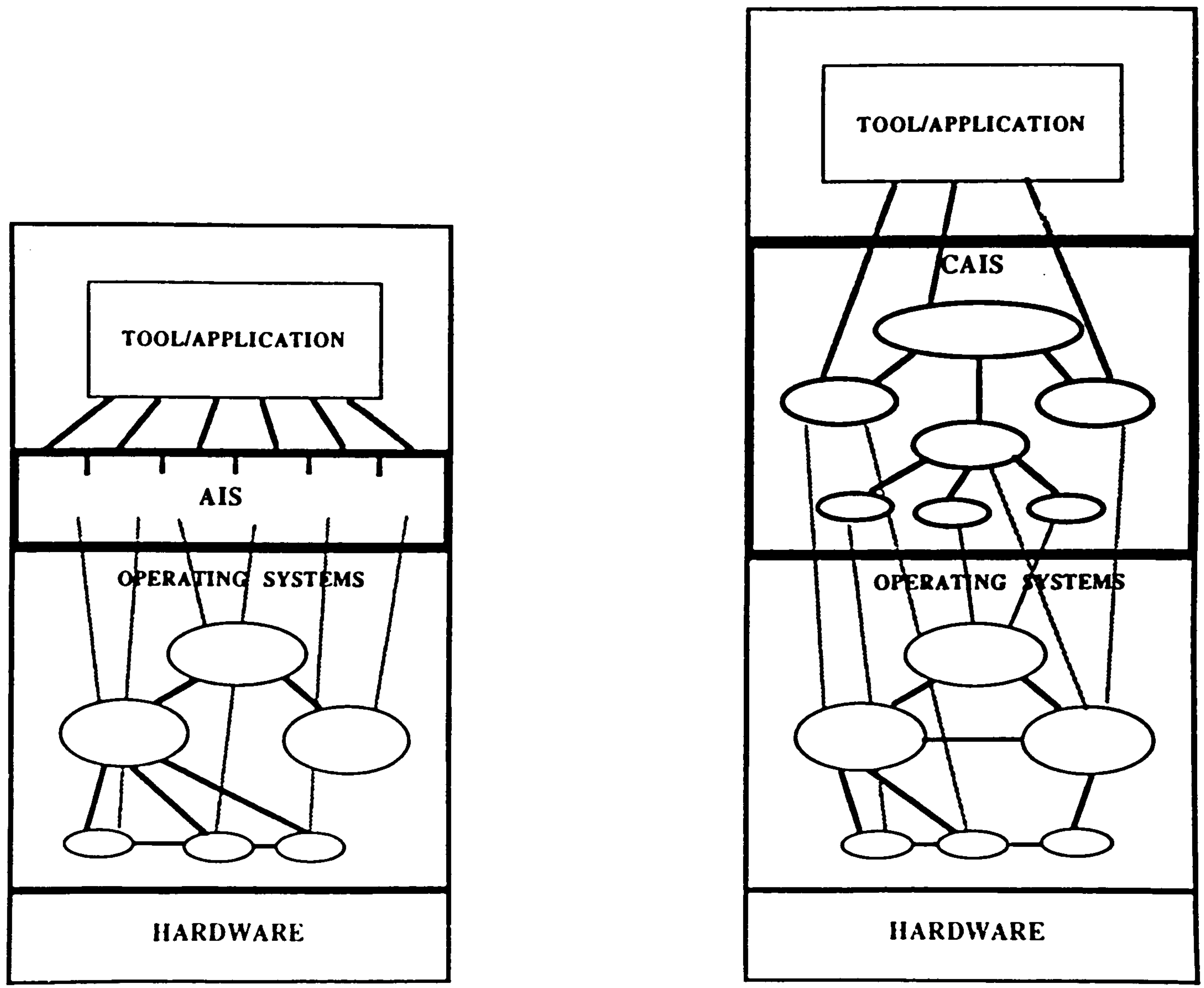

FIGURE 5: CAIS/AIS IMPLEMENTATION DIFFERENCES

EXPERIENCE TO DATE

The APCE is currently available on six different computer systems: VAX/VMS™[4], ROLM/AOS.VS, IBM/MVS™[5] and VM, and Intel 310 with XENIX™[6]. APCE processes can be distributed to the Macintosh™[7] and soon to the IBM PC. The rehosting process for the AIS takes approximately 2 calendar months for a mainframe and 1 month for a mini- or micro-computer. Figure 6, Current AIS Rehosts, illustrates the current systems the APCE is available on and the time it took to accomplish this, both in months and staff months.

APCE transportability has been attained using the AIS and a 'C' compiler. All APCE framework applications were designed using Ada™ PDL and implemented in 'C'. This was done because the Ada compilers were not available on all the hosts targeted for the APCE. The use of 'C' has not been without problems. Current implementations are using five (5) different 'C' compilers and as each new compiler has been introduced a 'C' subset has been defined. All APCE applications must be normalized to any new subset. This has entailed a five to ten percent code modification for each new subset. However, all new applications use the subset and are completely transportable. Because PRC must validate each 'C' compiler used for APCE code, the APCE will be recoded in Ada™ when validated compilers are available.

CONCLUSIONS

The APCE has the advantage that it can be installed in an existing configuration with minimal distruption of the current way of doing business. It provides a clear transition path into a better disciplined engineering process and allows new advances in automated tools to be incorporated. It does not, however, shield the users from a need to understand the native operating system or tool command language. This is not viewed as a disadvantage at this time since standardization of these features does not seem to be possible. Premature standardization of these features by an environment may ensure its technical obsolescence or, at best, enforce a delay while new tools are rewritten or rehosted. Such standardization is also not possible for a software house which works with a wide client base with widely differing requirements and standards for their software development and maintenance projects.

The APCE also does not provide the tight integration of tools. The user is still responsible for ensuring that the output of one tool is suitably modified to be acceptable as input for the next. This is one of the areas in which future work needs to be done to relieve the users of the more clerical types of work.

---

[4]VMS™ is a registered trademark of Digital Equipment Corporation.

[5]IBM/MVS™ is a registered trademark of International Business Machines, Inc.

[6]XENIX™ is a registered trademark of Microsoft Corporation.

[7]Macintoch™ is a registered trademark of Apple Computer, Inc.

REHOST		SCHEDULE TIME	STAFF MONTHS
Rolm MSE 800	AOS/VS	1.2 Months	4 Staff Months
INTEL 310	XENIX	1.5 Months	4.5 Staff Months
IBM	MVS	2.5 Months	12 Staff Months
IBM	VM/CMS	5 Months	13 Staff Months
Macintoch		1.5 Months	1 Staff Month

FIGURE 6: CURRENT AIS REHOST

The APCE framework provides significant advantages and can be used by a project without new hardware or significant retooling. It provides an immediate benefit without locking out future advances in software tools and techniques by managing the process and products rather than focusing on tools. The APCE provides a different approach to the software engineering environment problem.

PRC has been successful in rehosting the APCE to six different operating systems, with 4 different file management/database management systems that use 2 different sets of communication services without affecting the APCE applications. Since these different APCE instances can exchange project data and any APCE application is transportable between APCE instances, **the AIS attains true application independence.**

The benefits of using **an AIS like interface opens the options for the Space Station Software Support Environment (SSE)** configurations. **No longer constrained to only hardware independence** by operating system transportability; now a truly heterogeneous SSE can be configured. This environment will be **able to take advantage of all the required technology** while maintaining a consistent single environment through the SSE applications (tools and framework). The SSE will be **truly evolvable** since host services are divorced from the SSE itself therefore allowing new services (O/S, DBMS, communication and hardware) to be introduced and obsolete services to be retired **without disruption to operations.**