Open Systems Software Standards in Concurrent Engineering

D. Richard Kuhn
William J. Majurski
Wayne McCoy
Fritz Schulz

National Institute of Standards and Technology
Technology Administration
U.S. Dept. of Commerce
Gaithersburg, Maryland 20899

Reprinted from

Advances in Control and Dynamic Systems
Volume 62

C.T. Leondes, Editor
Academic Press, Inc.
1994

Open Systems Software Standards in Concurrent Engineering

D. Richard Kuhn
William J. Majurski
Wayne McCoy
Fritz Schulz

National Institute of Standards and Technology
Technology Administration
U.S. Dept. of Commerce
Gaithersburg, Maryland 20899

1 Introduction

The movement toward concurrent engineering comes at a time when the computer industry is undergoing a fundamental change in the way computers and software are developed and used. Products based on "open systems" standards – particularly the ISO Open System Interconnection (OSI), and IEEE POSIX – are beginning to replace reliance on proprietary computing platforms. Engineers who develop products that incorporate embedded computers and software are thus faced with the challenge of understanding two technological developments that will change the way they do their jobs: *concurrent engineering* and *open systems.*

This paper provides an introduction to open systems, explains important open system standards, and discusses how to use open system products effectively in concurrent engineering. The paper should be of value to engineers whose products depend on embedded software. The focus is on standards for software products that will be embedded in larger systems and products. Software development and integration are often the "weak links" in product development because of incompatibilities among embedded software components. A principal objective of open systems standards is to make systems integration easier.

2 Why Open Systems in Concurrent Engineering?

Concurrent engineering has been defined as

> ... a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. [Concurrent engineering] is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements. [1]

The emphasis is on developing a comprehensive understanding of all aspects of the product early in the lifecycle. When the product includes embedded software, this task can be particularly challenging. Today's software industry has been compared with the eighteenth century firearms industry, where each gunsmith built a complete product [2]. Eli Whitney, also inventor of the cotton gin, revolutionized the industry by introducing standards for firearm components.

> Each gunsmith focused on a single part, using sophisticated tools to optimize that task. This produced economies of scale that drove down manufacturing costs, and best of all, Whitney's customer, the government, quickly realized that the standards would allow parts to be interchanged, greatly simplifying their firearm repair problems [2].

Standard components for software have been advocated for at least two decades, but until recently only limited versions of such components, such as mathematical subroutine libraries, have been available. Barriers to more complex components have been both technical and economic.

Modern programming languages such as C++, Objective C, and Ada are helping to solve *technical* problems associated with component development, but a software component created today in a language such as C++ must still use system services (e.g., timers, database access, communications) that vary depending on the operating system, database, data communications, and other vendor specific functions. If a component makes extensive use of Digital's VMS operating system services, for example, then a different version of the component must be created for UNIX System V (see Figure 1). Open system standards will soon change this situation, thus making it *economically* practical to develop standard components.

Open system standards will be particularly effective in meeting the following requirements of the Defense Advanced Research Projects Agency Initiative in Concurrent Engineering (DICE), outlined by Sprague, Singh, and Wood [3]:

- *High commercial content*: use of commercial, off-the-shelf software, rather than custom-developed software.
- *Standards strategy*: migration path to accommodate changes in standards.
- *Commercialization strategy*: commercialization of DICE-developed modules.
- *Incremental usability*: implementation of the DICE architecture in stages, each of which provides a usable environment.
- *Low maintenance and training costs*: a concurrent engineering environment that is cheaper to set up and maintain than existing environments.
- *High user acceptance and satisfaction*: short learning curves and ease of use.

Although open systems address a broad range of needs for concurrent engineering, this paper will concentrate on characteristics of open systems and software components that can be used in meeting these important requirements. The remainder of this section explains the principal concepts of open systems; sections 3 and 4 describe some important open system standards; section 5 discusses the use of open systems in concurrent engineering, with emphasis on the DICE requirements listed above; section 6 concludes with a summary and discussion of directions for future research.

2.1 Open System Standards

In short, an *open system standard* is an *interface specification* – a specification that describes services provided by a software product – to which

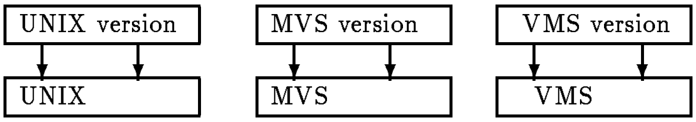

Figure 1: Application Software Today

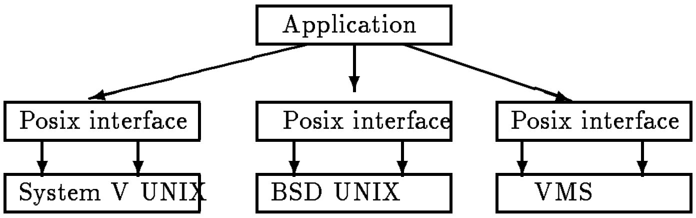

Figure 2: Open System Approach

any vendor can build products. There are two important points. First, the specification is available to any vendor and evolves through a consensus process that is open to the entire industry. Second, the specification defines only an interface, so different vendors can provide the standard interface on their proprietary operating systems (see Figure 2).

Open system standards will make it possible to develop standard software components that can be implemented on a wide variety of hardware, making a software components industry economically practical. But, open system standards do not solve all problems associated with building interchangeable software components. Software designers need to understand the capabilities and limitations of software standards, and how to deal with these limitations. This article describes important open system standards and explains how they can be used to build portable, interoperable application software components.

2.2 Interoperability and Portability

There are two important aspects to open systems: *interoperability* and *portability* [4]. Interoperability refers to the capability for applications running on different computers to exchange information and operate co-operatively using this information. Portability refers to the capability for software to run on different types of hardware. Portability can further be broken down into binary portability and source code portability. Binary portability makes it possible to move an executable copy of a program from one machine to another. Source code portability requires a program to be recompiled when moving from one machine to another. The development of portable application software components depends on portability standards. Interoperability standards are necessary but not sufficient for a complete open systems environment. Software systems that are built on standards for portability and interoperability are called *open systems.*

A good example of interoperability is provided by the X Window System [5] protocol, which specifies how graphics primitives can be communicated between an application program and graphics software running on a workstation. An X Window application running on an IBM workstation can interact, for instance, with a user sitting at a Sun workstation. The ISO Open Systems Interconnection (OSI) standards [6] [7] also promote interoperability. The OSI reference model defines a structure, or *reference model* for data communication standards. The reference model defines seven layers of communications system components, from hardware at the bottom to application software at the top. The model describes how components communicate with each other; i.e., it is a model for interoperability. Open system standards for application portability do not "fit in" to the OSI reference model; they are, however, complementary to data communication standards. Communication standards define communication services, but open system applications require a standard way to use those services.

Binary portability specifications are designed to provide software portability at the object code level. For example, the IBM PC hardware interface can be regarded as a *de facto* standard for binary portability. Executable copies of software can run on PC clones from many different manufacturers. Another example is the Application Binary Interface for systems based on the Sun SPARC processor. This specification for workstations makes it possible to move executable programs between different makes of workstation as easily as programs can be moved between different IBM PC clones. Binary portability is more difficult to achieve than source code portability, because it places constraints on hardware. Standards efforts have concentrated on developing interfaces for source code.

Open system standards for source code portability define interfaces available to application programs for services such as timing functions, security features, database access and many other essential functions. Standards could be defined by cooperatively developed source code, but within IEEE and other organizations the preferred approach has been to specify interfaces and let vendors develop competing implementations. Thus, two different operating systems may provide the same services, but one may have better performance or fault tolerance characteristics than the other. The application program interface may be specified in terms of a set of procedure calls for a particular programming language, or a language-independent specification may be accompanied by procedure calls for one or more programming languages.

The best-known standards for operating system services are the POSIX standards being defined by the IEEE Technical Committee on Operating Systems (TCOS). (The acronym POSIX is derived from Portable Operating System Interface, with an "X" to denote its UNIX origin.) Beginning with the POSIX System Application Program Interface (or kernel) standard (IEEE 1003.1-1988), IEEE has been developing a comprehensive set of standards for application portability. In May 1992, the POSIX effort comprised 20 working groups developing 34 projects.

The POSIX efforts (1003.x) have been supplemented with projects to develop standards for application interfaces to services such as windows (1201.1) and X.400 message handling (1224), that are useful on non-POSIX systems. Other open system standards have been developed through ANSI, ISO, and other organizations. Many of these other specifications have been combined with the developing IEEE standards to define an open systems environment using the POSIX interface standards as the basis.

3 The POSIX Open Systems Environment

No single standard provides all the functionality needed by a modern computing environment. Portability and interoperability require a comprehensive set of standards. The POSIX Open Systems Environment (OSE) being

$$
\begin{array}{c}
\boxed{\text{Application Software}} \\
\Uparrow \\
\text{Application Program Interface (API)} \\
\Downarrow \\
\boxed{\text{Application Platform}} \\
\Uparrow \\
\text{External Environment Interface (EEI)} \\
\Downarrow \\
\boxed{\text{External Environment}}
\end{array}
$$

Figure 3: POSIX OSE Reference Model

put together by IEEE TCOS working group P1003.0 [8] provides a standard set of interfaces to information system building blocks, covering both portability and interoperability standards. Not all of the specifications in the POSIX OSE are IEEE POSIX (1003.x) standards. POSIX functions serve as a basis, supplemented by other applicable open systems standards.

Two types of standard interfaces are specified in the POSIX OSE: the application program interface (API) and external environment interface (EEI). The POSIX OSE Reference Model, shown in Figure 3, shows the relationship of these interfaces to the other parts of the computing environment.

The External Environment refers to external entities with which the application platform exchanges information, including both the human end user, hardcopy documents, and physical devices such as video displays, disks, printers, and networks. External environment interfaces mainly provide for interoperability. EEI standards take the form of communication protocols, record and document formats, and display formats. The application program interfaces in the POSIX OSE are source code interfaces, generally in the form of programming language procedure calls, to the application platform, which is the operating system and hardware. By specifying a standard set of procedure calls, an API provides source code portability. To explore how standards can be used in constructing open software it is useful to examine the POSIX OSE application program interfaces in some detail.

The POSIX OSE API contains four categories of interfaces: System Services, Communications Services, Information Services, and Human Computer Interaction Services. A typical computing environment will require some, but not all, of the standards contained in each of these four categories. A fifth category, Domain Services, is provided for special purpose environments, such as transaction processing. Each of the four main categories includes a collection of applicable standards, although some of the standards provide different services than others within the same category. The POSIX OSE guide also includes specifications such as UNIX System V Interface Definition and OSF/1. These are not formal standards but proprietary specifications which include standard functionality as a subset.

3.1 Fundamental System Services

This category includes both language services and operating system services. Language services are the functions typically provided by programming languages such as C, Fortran, Pascal and others. Operating system services are the services normally associated with an operating system or executive such as UNIX or VMS.

The language service area gives standard specifications for such programming languages as Ada, BASIC, C, C++, and Pascal. To make other services in the OSE accessible from applications programs, language bindings are needed for one or more of these languages. The POSIX kernel standard (1003.1), originally defined using C, has Fortran (1003.9) and Ada (1003.5) language bindings also. Standard interfaces to all of the other services have not necessarily been defined for all of the languages in the POSIX OSE. The most common language for POSIX interfaces is C. In addition to providing a way to call system services, the languages themselves provide functions for programs, including mathematical functions, data definition and representation, error handling, I/O operations, and program control logic.

Operating system services are used to control the resources of the computer system. They are the functions provided by an operating system or executive program, such as UNIX, VMS, Multics, and many others. The major categories of operating system services in the POSIX OSE are process management; task management; environment services; process communication and synchronization; input/output; file management; event, error and exception management; time services; memory management; logical

Language	Standard (or working group)	Status
Ada	ISO 8652:1987	complete
BASIC	ISO 6373:1984	complete
C	ISO 9989:1990	complete
C++	ANSI X3J16	in progress
COBOL	ISO 1989:1985	complete
FORTRAN	ISO 1539:1980	complete
Pascal	ISO 7185:1983	complete
PL/1	ANSI X3.74-1981	complete

Table I. POSIX OSE Language Standards

naming; system initialization, reinitialization, and shutdown. Standards in the OSE providing these functions include POSIX Shell and Utilities (1003.2), Realtime (1003.4), and the Microprocessor Operating System Interface (IEEE 855).

3.2 Communications Services

Communication services make communication possible for application programs running on machines connected via a network. They include services for file transfer, namespace and directory services, network file access, remote procedure call, protocol-independent network access, and data representation. Both Application Program Interface and External Environment Interface functions are included in this area.

The EEI functions provide interoperability and include standards for physical connections, network protocols and formats, and distributed system services. The ISO Open System Interconnection standards are the most significant component of the EEI services.

The interface to the interoperability functions is through the standard APIs, such as the Protocol Independent Interface (1003.12) and the remote procedure call interface being developed by ANSI X3T5.5. The communications services component of the POSIX OSE is still evolving; other standard services may be added later.

System	Standard	Status
System API (Kernel)	ISO/IEC 9945-1:1990 (same as IEEE 1003.1)	complete
Shell and Utilities	IEEE 1003.2	complete
User Portability Ext.	IEEE 1003.2a	complete
Realtime Extension	IEEE 1003.4	in progress
Threads Extension	IEEE 1003.4a	in progress
Security Interface	IEEE 1003.6	in progress
Administrative Service Interface	IEEE 1003.7	in progress
Microprocessor Operating System Interface (MOSI)	IEEE 855	complete
Common Ada Programming Support Environment Interface Set	MIL-STD-1838A	complete

Table II. POSIX OSE Operating System Service API Standards

Function	Standard	Status
Protocol Independent Interface	IEEE 1003.12	in progress
File Transfer Access Method	IEEE 1238.1	in progress
Distributed File Access	IEEE 1003.8	in progress
X.400 Interface	IEEE 1224	in progress
Remote Procedure Call	ANSI X3T5.5	in progress
Remote Data Access	ANSI X3T2	in progress

Table III. POSIX OSE Communication Service API Standards

3.3 Information Services

Information services include Database Services, which provide the capability to store and retrieve data from long term storage, and Data Interchange Services to exchange data between systems.

Data Interchange Services include data description protocols, character sets, and data format protocols. Data description protocols provide a standard means of associating a name with individual data elements. Data format protocols add attributes that describe the physical characteristics of the data. Among the Data Interchange Services are the Standard Generalized Markup Language (ISO 8879:1986) that can be used to define the layout and structure of a document, the Electronic Data Interchange standards being developed in ANSI X12 for Data Format Protocols, the Computer Graphics Metafile (ANSI X3.122-1986) which provides a standard means for storage and exchange of computer graphics, and the X.400 Message Handling System (ISO/CCITT X.400-1984,1988) which defines message protocols for electronic mail. There are currently no API standards in the Data Interchange area.

Database Services are the functions associated with database management systems. These include: data definition and manipulation - the ability to create, update, delete records, fields, or tables; data access - the ability to retrieve data based on complex search conditions; and data integrity - locking of data items, transaction control, and synchronous writes. Application programs use database services extensively, and the APIs in the Information Services area include the Structured Query Language (ISO 9075:1982) and Network Data Language (ISO 8907:1987).

3.4 Human/Computer Interaction Services

Human/computer interaction services provide functions for communication between user and computer, using the window and mouse style of interaction popularized by the Apple Macintosh.

Applicable external environment interface standards will include the X Window protocol, which specifies the format and meaning of messages between application program and display device, and human factors standards such as the User-System Interfaces and Symbols standard being developed within ISO-IEC/JTC1/SC18/WG19, and the ISO 9241 standards for Ergonomics of Visual Display Terminals. The "drivability" Recommended

Function	Standard	Status
Network Data Language NDL	ISO 8907:1987	complete
Structured Query Language SQL	ISO 9075:1982	complete
Information Resource Dictionary System	ANSI X3.138-1988	complete
Information Resource Dictionary System	ISO DP 8800 N2132:1988	in progress

Table IV. POSIX OSE Information Service API Standards

Function	Standard	Status
Layered API	IEEE 1201.1	in progress
Xlib, Xt	FIPS 158	complete

Table V. POSIX OSE Human/Computer Interaction Service API Standards

Practice being developed in IEEE 1201.2 will also be important within the EEI. It will recommend a set of window system behaviors designed to make working with different window systems as easy as driving different makes of automobiles.

API standards in this area are still being defined. The X Window system interfaces in Federal Information Processing Standard 158 provide a low level interface to graphics and window functions. Most designers prefer to use higher level functions such as those provided by Sun's Open Look or the OSF Motif System for window-based applications. Each high level function corresponds to an object on the display screen, such as a scrollbar or button. When completed, the IEEE 1201.1 Uniform API standard will probably be the primary high level API for the Human/Computer Interaction Service component. IEEE 1201.1 is intended to be a set of window system function calls that can be used with any system that provides the necessary services to create and manipulate menus, buttons, scrollbars, graphics and other common features of window-based interfaces.

4 Standard Communication Services

Communications is a service that is key to the success of concurrent engineering. It is the foundation for enabling engineers, designers, developers and production teams and systems to collaborate and coordinate all the efforts involved in developing and manufacturing a product. A true concurrent engineering environment will involve sharing of data, graphics, text, software, processing, manufacturing process control and display among groups of participants in the concurrent engineering process. If this is to be efficient, it must be supported by robust, speedy and flexible communications. This section discusses two open communications standards that will greatly enhance the ability to achieve the goals of concurrent engineering.

4.1 Open Systems Interconnection

The Open Systems Interconnection (OSI) Reference Model is an International Standard, ISO 7498, developed through collaboration of the Organization for International Standardization (ISO) and the International Telephony and Telegraphy Consultative Committee (CCITT). This seven-layer model abstractly defines the services and functions required to carry out data communications. A fifteen year effort has resulted in public specifications for protocol functions at all layers. In the United States, implementation agreements for these standards have been produced by the Open Systems Environment Implementors' Workshop (OIW). The National Institute of Standards and Technology (NIST) has issued Federal Information Processing Standard 146-1, the Government OSI Profile (GOSIP), which is based on the international standards and the implementation agreements. Of particular interest to concurrent engineering will be the set of agreements culminating in the Industry/Government Open System Specifications (IGOSS) being developed by the OIW. IGOSS is expected to be made a standard profile for OSI communications to replace GOSIP. IGOSS should be delivered to the public in 1993 and result in products by 1994.

4.2 ISDN

In many ways, the functionality and services of Integrated Services Digital Networks (ISDNs) [9] will more nearly meet the technical needs of concurrent engineering than will those provided through the current OSI definitions, particularly with regard to Broadband ISDN. One of the reasons for this is that the existing, or emerging, OSI standards address primarily only conventional data communications. The communications requirements for concurrent engineering are quite different in many ways from those of ordinary business. Concurrent engineering can make great use of functions that permit engineers and designers remote from one another to collaborate in a conference featuring voice, data, graphics and other images, and video, all displayed at each collaborator's workstation and carried on a single communications interface. ISDN provides this capability now. Current OSI definitions include ISDN as a subnetwork service but do not consider the full potential of the ISDN services. It is reasonable to expect, however, that at some point in the future OSI will include full use of ISDN.

The standards for ISDN have emerged from the International Telephony and Telegraphy Consultative Committee (CCITT) and are further refined by the American National Standards Institute ASC T1.S1. The functional model for ISDN follows that of the OSI Reference Model. However, as noted above, ISDN is not part of the suite of OSI data communications protocols. In North America, implementation agreements and conformance tests for ISDN are being produced by the North American ISDN Users' Forum (NIUF) [10]. NIST has proposed a Federal Information Processing Standard based upon these implementor agreements. The CCITT has accepted the conformance tests produced by the NIUF for use in testing implementations. An implementation, closely correlated with the implementor agreements and referred to as National ISDN-1, has been developed by Bellcore and is currently being deployed by telephone companies in the United States and Canada. Similar efforts are going on in Europe and Japan. It is expected that the Regional Bell Operating Companies (RBOCs) will have ISDN deployed in more than 95% of their telephone central offices by 1996. In this respect, ISDN is a true "open system".

Although ISDN would appear to be "just another network service", it offers far more capability than most of the data networks that are now in place. For example, ISDN includes X.25 as one of the services provided. It also provides: circuit switching; multiple information channels per physical interface; independence of user protocols on principal information channels ("bearer" or B- channels); separate, out-of-band signaling channels; selection of service and bandwidth on a per-call basis; supplementary services, including conferencing of data, video or voice, and call management; guaranteed bandwidth to users; exceptionally low error rates; integration of all services through a single physical interface. By about 2000, Broadband ISDN (B-ISDN) should be widely available; some RBOCs are installing it now on a limited basis. Aggregate data rates available per user of B-ISDN will exceed 1 Gb/s by 2000.

Currently, most user software interfaces for accessing ISDN are proprietary. However, the NIUF is developing an interface intended for standardization in the U.S. and Canada. The NIUF Application Software Interface (ASI), despite its name, is not an OSI Layer 7 interface. The word "application" in ISDN contexts refers to usage of the ISDN services and not in general to end-user functions, as is the case in other contexts. The ASI, which has no end-to-end significance, is intended to fit into the OSI Reference Model between Layers 3 and 4, although the operational significance of this with regard to OSI Transport and Network functions has not yet been defined. Currently, the ASI is defined for operation in two different operating system environments: DOS and UNIX/POSIX. Definitions for other environments may be developed later. When the ASI is fully accepted by the NIUF, it will be put into the standards process, likely through the canvas process.

Wide usage of ISDN will likely cause more attention to be paid to the other standards described in this paper, since the interchange of information, software and processing that is possible using ISDN should accelerate the original reasons for the development of those standards.

5 Other Frameworks

To be useful in a concurrent engineering environment, technologies, both standard and proprietary, must be integrated into a coherent unit. There are three new challenges to be faced:

- In the past, system integration has been an issue for system vendors and integrators. With the advent of open systems, end user organizations are also placed in the position of integrating components from multiple vendors into working systems.

- As demonstrated earlier in this chapter by the number and diversity of work on standards for component technologies, there is an unprecedented number of new technologies being readied for the marketplace.

- Most existing commercial system architectures are limited in their ability to support the future needs of the industry. The upcoming generation of operating systems will offer support for integrated networking and distributed systems and offer integration technologies: services and techniques for developing software components that are designed for later integration. System integration itself has become the target for new technology development.

In the previous generation of computing equipment, users chose a system vendor and that vendor supplied almost everything: hardware, software, and maintenance. Discussion of architectural issues of the system occurred primarily within the vendor's development sites. Interoperability of systems within the vendor's product line was a concern for the vendor's engineers.

To make these integration technologies work, some aspects of system architecture and design must be discussed in an open, non-proprietary forum. For aspects of systems that influence integration, dealing with system design and implementation would mask the strategies of interest. Instead, discussion of frameworks centers on abstract properties of systems that are necessary to guide system developers.

A framework is a basic structure or set of architectural principles for constructing a system. In the current generation of systems being developed this typically includes an object model, sets of services to be provided by elements of the object model, and necessary properties of objects.

In the open system environment interoperability must be negotiated across corporate boundaries and involving a greater base of customer requirements. This negotiation is typically stratified into three layers:

- General abstract concepts and models with input from academic and industry researchers.

- Concrete models and reference implementations from industry consortia.

- Products from individual companies.

The need for interoperability has led vendors to work together to develop the necessary integration technologies. The traditional standards development process leaves some aspects of a standard unspecified for many reasons. Such standards are then followed by the development of application profiles, standards applied to specific environments and needs in which the specifications can be tightened to remove ambiguities and missing elements. Next, implementors cooperate in conformance testing to ensure interoperability of their implementations.

Most standards work is conducted within a single domain; database, network protocols, and languages are examples. Implementations are then made within a vendor's operating system or other product architecture, the standard implementation being the only new piece. Framework standards, whose goal is integration of these uni-domain standards, force changes in basic system architecture. Far more risk is therefore involved in adopting framework standards. In many instances consortia are necessary to pool resources in development beyond standards work. The resulting reference implementations are the basis for products. Interoperability is far easier to obtain and maintain starting from a reference implementation.

Both OSI and the POSIX OSE can be considered frameworks, but other important efforts are being conducted in this area as well. This section will discuss four other frameworks briefly, two abstract and two concrete. The Basic Reference Model for Open Distributed Processing (RM-ODP) [11] offers abstract guidance in the issues of developing an integrated open system. The X Window System [5] is a commercially available multi-vendor system that embraces the framework concept in its architecture. The X Window System will be used to illustrate some of the abstract principles of RM-ODP and its architecture. The Reference Model for Frameworks of Software Engineering Environments [12] applies a similar set of abstractions to the development of software engineering environments. The last framework, results of a consortium called the Object Management Group [13] [14], illustrates the steps in making this kind of technology viable in the marketplace.

5.1 X Window System

The X Window System offers a framework for a class of application programs. This framework includes a programming model and a communications model, as well as a model for windows applications. This underlying framework or architecture creates the foundation supporting programming APIs. This gives the programmer a context for designing applications and later gives the integrator a context for combining applications or parts of applications into a finished system.

This framework was established and published so that display servers (display drivers) from multiple vendors are interchangeable. Vastly more important, window clients (application programs) are constructed with standardized components resulting in interoperability and reuse across vendor specific environments. This class of window application programs is also capable of communicating among themselves using the display server as a communications relay. The result is that servers and clients become "pluggable" modules to be used in creating systems within the framework.

The architecture and initial implementations of the X Window System was originally developed at the Massachusetts Institute of Technology. The maintenance/evolution was taken over by a consortium and the source code continues to be available for a nominal charge. The control of this technology by a central group and the universal distribution of the source code has made it possible to advance the system while simplifying issues of interoperability.

5.2 Basic Reference Model for Open Distribute Processing (RM-ODP)

The development of this abstract framework is the target of work in an ISO working group (RM-ODP) (ISO 10746, Recommendations X.901 - X.905) [11]. The objective of this group is defined as follows:

> The rapid growth of distributed processing has lead to a need for a co-ordinating framework for the standardization of Open Distributed Processing (ODP). This Reference Model of ODP provides such a framework. It creates an architecture within which support of distribution, interworking, interoperability and portability can be integrated [11].

More specifically, this ISO work is about standardization of distributed processing architecture and external linkage, so that coherent systems can be constructed from heterogeneous and re-usable components, and multiple processing systems.

A set of generalized user requirements for distributed systems is codified as a guide for system designers. These range from issues of availability and reliability to integration, performance, and transparency. The discussion of transparency is of interest in that it describes the level of flexibility users seek in systems. The requirements stipulate:

1) access transparency - hiding the use of communications;

2) location transparency - hiding the topological details;

3) replication transparency - hiding the effect of multiple copies;

4) concurrency transparency - hiding the effect of parallel execution;

5) migration transparency - hiding the heterogeneity of system components to enable the migration of system functions between components;

6) failure transparency - hiding the occurrence of errors in system components and communications;

7) liveness transparency - hiding the transfer of objects between capsules and storage objects;

8) configuration transparency - hiding the configuration and reconfiguration of components within systems.

RM-ODP creates a framework of abstractions to position services relative to one another, to guide the selection of appropriate models of services, and to place boundaries on ODP. This framework is divided into viewpoints of system (enterprise, information, computational, engineering, and technology) and aspects (process, storage, user access, communication, identification/naming, management, and security).

5.2.1 From RM-ODP to X Window System

As described above, RM-ODP details eight areas of transparency which are desirable in system architectures. These abstract issues can be related to features in the X Window System. The features described here can all be achieved by following suggested coding guidelines.

Access transparency - hiding the use of communications: X clients can operate with a local display server (same workstation) or across a network transparently (to the developer of the client and the user of the client).

Location transparency - hiding the topological details: X (BSD/UNIX implementation) relies on the underlying socket mechanism for managing TCP/IP connections. The developer/user specifies a location identifier which details the display host (using local network conventions) and a display on that host (by index number).

Configuration transparency - hiding the configuration and reconfiguration of components within systems: X employs, within its server, a database of attributes (parameters) applicable to systems, applications, windows, elements of windows etc. This database is managed local to the display server and controls the configuration of reusable client programs.

The other features listed in RM-ODP: replication, concurrency, migration, failure, and liveness transparencies are not applicable or not available within this framework. Any further discussion here would require analysis of the basic scope of the X environment.

5.3 Reference Model for Frameworks of Software Engineering Environments (RM-SEE)

Another reference model of interest is the "Reference Model for Frameworks of Software Engineering Environments" [12] developed jointly by NIST and the European Computer Manufacturers Association (ECMA). This reference model serves as a conceptual basis for describing and comparing software engineering environments which provide automated support of the engineering of software systems and the management of the software process.

This reference model defines a set of services which model the functional capabilities of an SEE framework. These services are grouped into object management, process management, user interface, communications, and tools supported by the framework. Like the RM-ODP reference model, it is based on an object model.

As we have mentioned many times already, integration is the most fundamental issue in the use of standards. RM-SEE offers a definition of integration. Integration can be described as a set of services. Using a common object management system with a common schema permits tools to share objects. Common presentation characteristics in the user interface allow for building common "look and feel" features across tools. Process management services and communication services are needed for tools to communicate with other tools. This reference model, when compared with RM-ODP, concentrates less on object systems, formal methods, and issues of distribution and more on issues relating to their use in constructing tools, user interfaces, and design repositories.

Having common services allows but does not force integration. Having a common object management system and scheme permits but does not force tool builders to use it. An enforcement policy is required so that various platform, framework, and tool builders use the various integration services effectively. This may either be implemented as enforcement services in the framework (resulting in restricted interfaces available to service/tool implementers) or as a style guide of expected practices by tool builders.

Integration is the relationship among (several) components in an SEE. This may be between tool and framework, tool and user interface, tool and tool, tool and object management system. There are several perspectives in describing integration. From the user's perspective, an integrated SEE provides a common view of the system. The entire environment operates as one coherent tool. From the tool developer's perspective, it provides a consistent interface for building tools. The functions needed to interact with the object management system, process management, user interface and other services should be clearly specified. Tools should be able to pass information to other tools in an easy manner. In general, integration has been identified in several areas: data integration, control integration, presentation integration, process integration, and more general framework integration.

5.4 Object Management Architecture

This section describes the work of an industrial consortium and not a standards body. We include this as an example of the process of constructing an industrial strength computing environment in today's "standards driven" environment in which consensus for the sake of interoperability is crucial.

OMG has published two key documents. The first, "Object Management Architecture Guide 1.0" [13], defines an abstract object model. This model defines the theoretical model that proposed technology (proposed to the consortium) should conform to or support and a glossary of terms to provide some degree of uniformity of concepts. Vendors are expected to propose technologies positioned within the reference model components that when taken together achieve a set of stated technical objectives.

The second document, "The Common Object Request Broker: Architecture and Specification" (CORBA) [14], defines a concrete object model. A concrete object model may differ from the abstract object model in several ways:

1) It may elaborate the abstract object model by making it more specific, for example, by defining the form of request parameters or the language used to specify types.

2) It may populate the model by introducing specific instances of entities defined by the model, for example, specific objects, specific operations, or specific types.

3)It may restrict the model by eliminating entities or placing additional restrictions on their use.

Among other things CORBA defines: specific object semantics such as requests, creation and destruction, types, and attributes; the structure of a client and an object implementation; an interface definition language; and a C language mapping. These necessary elements help realize an abstract framework for computing.

5.5 Common Aspects of Frameworks

These frameworks have much in common. They all rely on an object model as an architectural base and use a relatively common vocabulary for elements of technology. In the abstract, the general organization of services is also relatively common and hopefully will foster similar thinking by architects of the generalized system components.

These integrated environments, the target of open systems, must rely on common architectural approaches, frameworks, if integration in open systems is to succeed. Specific services aimed at integration must be made available in the resulting products.

6 Using Open System Standards in Concurrent Engineering

The primary goal of open systems is to make application software more portable and interoperable. Portability and interoperability, however, require more than simply the use of standards. It is important to take the special characteristics of software standards into account when designing open applications. This section discusses these characteristics, provides an architecture for standard software components, and explains how these standard components can be used in concurrent engineering.

6.1 Potential Pitfalls of Software Standards

Standards reflect industry consensus at a given point in time. As innovations are made and spread among vendors, standards must change to reflect what has become new common practice. To accommodate this process, the IEEE and other standards bodies schedule standards for periodic revision. A typical revision period is five years. If an application depends on the use of n standard components, there is a potential for changing the application to meet changes in the standards $n/5$ times a year. A large application might use 10 or more standards. Since most large applications take several years to complete, developers can expect some of the standards they are using to be revised during the development period. Standards revisions are also a factor in maintenance. For an organization with k applications, $kn/5$ change efforts may be needed just to keep up with evolving standards. While the changes needed may be small, as most changes to standards are to provide new functions rather than modify existing ones, an effort will be required to review applications to determine whether changes are needed or not.

The varying stages of completion or revision of standards can add to integration difficulties during development. A system may need services that are not available from any existing operating system, either because a standard for the service interface is not completed, or because the existing standard is about to be revised and changes are expected in revision. Since standards necessarily lag the state of the art, some applications will need services that cannot be provided by any standard interfaces. Different systems may have different ways of providing a service.

6.2 An Architecture for Standard Software Components

Given some of the shortcomings of open system standards, what is the most effective way of using them to achieve applications portability?

One approach is to build components that provide services specific to an industry or an individual organization, resulting in a hierarchy of services: generic system services provided by standards such as the POSIX kernel, industry-specific services provided by components built on the system services, and organization-specific services provided by components built on the industry-specific services and the generic system services. Application

Application Programs

$\downarrow$

Proprietary Software and Hardware

Figure 4: Conventional Architecture

programs for end-users can be built to use the application program interfaces provided by the hierarchy of components.

A simple example of an organization specific interface might be a specification for a function that displays a company logo, department name, and time of day on a graphics terminal. A software component to provide the specified service uses operating system functions to obtain the time of day, bitmap for the logo, and department information from a database. Many of the organization's application programs use the same service, and the application programs may run on many different brands of computer. The application programs call these APIs, rather than calling operating system services directly. This approach is sometimes used today to deal with system dependencies, and it will still be necessary when open system standards are used.

It is also possible to specify an API for a particular industry. For example, ISDN service standards are being defined for the telephone industry, banking, retailing and other industries. NIST and the Interactive Multimedia Association are developing an API for multimedia services to be used in computer-aided training systems. Components providing the services specified in the API can be built on standard interfaces. Because they are built using open system standards, the components can be ported to a wide variety of hardware at low cost. Without open system standards, multiple versions might need to be developed to accommodate differences in file system structure, network interfaces, and other system services. Component development becomes more attractive for software developers because it is not necessary to produce a separate version for different operating systems.

Application architecture is usually as pictured in Figure 4. With open system standards, a model such as that shown in Figure 5 is more useful. Arrows represent use of a component's services by the component at the tail of the arrow. Application programs use functions provided by an

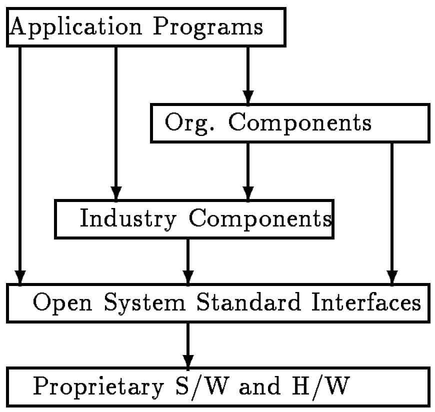

Figure 5: Open System Component Architecture

organization-specific API, by an industry standard API, or they may use standard functions directly. An organization-specific API is built using services provided by industry standard APIs or by standard open system services such as POSIX. Components supplying industry specific services can be built using the standard services and marketed for use on many different hardware platforms.

Organization standard services isolate the application from the interface with industry standard system services. Modifications to accommodate changes in organization requirements are restricted to the organization standard service modules. The organization standard services can be compared with objects in the object-oriented programming paradigm. The difference is that an organization standard service is more general than a typical "object". Consider a video game example: an object might be a ship that is moved by the user, the organization standard services could be functions that calculate speed and heading from coordinates, while the standard functions used are provided by the compiler's math library. In this case the organization standard services are equivalent to a very general class of object that moves in a two-dimensional plane.

Industry standard system services thus provide independence from hardware and operating systems, while the organization standard services provide independence from system services. This helps insulate the application from proprietary operating system services as well as from changes in standards.

6.3 Meeting the DICE Requirements Using Open Systems

A number of considerations come into play when a product includes both hardware and software components; a class which includes the majority of products available today. The use of open systems for concurrent engineering has implications for both the engineering process, and the result of the process: the software component. How will open systems affect concurrent engineering? How can open systems help in meeting the needs of concurrent engineering? This section provides some preliminary answers to these questions by examining the impact of open systems on each of the DICE requirements introduced in Section 2.

- *High commercial content*: use of commercial, off-the-shelf software, rather than custom-developed software.

  The POSIX, OSI, and ISDN standards provide an extensive range of functions for use in developing application software of all types. With standard software functions available for nearly everything from security, to real-time, to fault tolerance, it will be possible to build almost any type of application using open system products.

  It is likely that no product will require all types of functions available from the many open system standards. A subset of standard functions is typically sufficient to meet product requirements. Such a subset is referred to as a "profile". Profiles for different types of applications, such as transaction processing, supercomputing, and multiprocessing, are being developed within the POSIX working groups. Profiles are incorporated in the Domain Services portion of the POSIX OSE.

  Industries and organizations may also have their own profiles, based on their unique needs. Grouping standard functions in profiles can make it easier and cheaper for vendors to provide software, just as automakers can offer "packages" of options for lower cost to the buyer than would be required if each option in the package were purchased separately.

- *Standards strategy*: migration path to accommodate changes in standards.

  As discussed in Sections 6.1, accommodating changes in standards can be difficult. The architecture outlined in Section 6.2 helps mitigate these problems by reducing the dependency of product software on generic standards.

- *Commercialization strategy*: commercialization of DICE-developed modules.

  The use of standards-based application platforms will result in major improvements in the ability to market tested and proven software components. If DICE-developed modules are based on open system platforms, they will have a broad base of potential customers, since computer vendors are moving toward open systems. The availability of common API specifications for a wide variety of application platforms greatly broadens the scope of reuse for software components. Each new application platform or area of application is an opportunity to avoid the tremendous cost of bringing a software component to a proven and stable state.

- *Incremental usability*: implementation of the DICE architecture in stages, each of which provides a usable environment.

  Every application platform in the manufacturing environment benefits from a range of procurement options. Each may be updated using new technology as it becomes available. Standardized APIs present a stable view of the underlying platform to the application programmer (and the software), and isolates them from the rapidly changing technology under the interface. This will allow revolutionary changes to occur within the platform, with the attendant benefits of greater performance and broader function. And all of this can happen with minimum change to source code as it migrates to newer technology.

- *Low maintenance and training costs*: a concurrent engineering environment that is cheaper to set up and maintain than existing environments.

  Standard API based software platforms are providing new options across the full range of manufacturing applications. These include the

CAD user using solid modeling packages to develop product descriptions, the programmer using a software development environment to produce software components of a product, and the processor to be included within the product.

- *High user acceptance and satisfaction*: short learning curves and ease of use.

  Open systems will enable better use of the pool of skilled programmers. Previously a programmer specialized in a certain application area and learned the programming languages and operating systems that had evolved for that specialized niche. These languages and operating systems saw limited use in other types of application. Programmers who switched specialties faced a substantial learning curve before they could contribute in other application areas. With open systems, the programmer will still need to develop an understanding of the new application area, but will need to learn fewer new operating systems software interfaces since familiar open system interfaces will be used across a broad range of application types.

The ability to test and validate software components is greatly enhanced by using open system components. Even the simplest software cannot be exhaustively tested under practical circumstances. The software production team has conflicting requirements to minimize the test sets to meet time constraints, and to maximize the test sets to provide greater assurance of quality and correctness. A balance must be struck. While the same may be said of many non-software product components, the test and quality assurance aspects of software are less well understood.

Stable, standards-based application program interfaces will provide the opportunity to build stable test concepts and technology. Tools and and simulation techniques are evolving to make this task much easier.

7 Conclusions and Future Directions

Open system standards will have a significant impact on both cost and competition in the computer industry. Software products can be produced more efficiently, because developers can produce a single version for the standard programming interface, rather than a different version for each vendor. Vendors will be able to compete for business that previously was denied to them because of users' dependence on their current vendor. Users can buy from different vendors without requiring major conversions of their internally developed software. Since software products can be developed more efficiently for a wider range of hardware, software components become more economically practical.

Real benefits can be expected from the use of open system standards, but the degree of success with software components built to conform to these standards is dependent on quality conformance testing. Standards for materials, weights, and measures can now be stated with remarkable precision; conformance to these standards can be determined with a correspondingly high level of confidence. Software standards are generally much less precise [15]. With few exceptions, software standards today use English to explain what the software is required to do. This can result in ambiguities in the standards that lead to incompatibilities in products, and inconsistencies between test suites. Without precise specifications, it is very difficult to measure conformance with a high degree of confidence.

Successful use of open system components in product development depends on making software more reliable and software development more predictable. To meet the needs of U.S. industry and government for quality software in the 1990s and beyond, research is needed to improve methods for precision in specification and testing of software systems.

7.1 Disclaimers

Use of specific products and companies as examples in this article does not indicate endorsement by NIST or by the U.S. Government, nor does such use imply that the products named are the best available for the stated purpose.

UNIX is a trademark of AT&T. VMS is a trademark of Digital Equipment Corp. MVS is a trademark of International Business Machines Corp.

References

[1] R.I. Winner J.P. Pennell H.E. Bertrand M.M.G. Slusarczuk. The role of concurrent engineering in weapons system acquisition. Technical Report R388, Institute for Defense Analysis, 1988.

[2] B.J. Cox. *Object Oriented Programming, An Evolutionary Approach.* Addison-Wesley, 1986.

[3] R.T. Wood R.A. Sprague, K.J. Singh. Concurrent engineering in product development. *IEEE Design and Test of Computers*, March 1991.

[4] G. Fisher. *Application Portability Profile.* National Institute of Standards and Technology, 1990.

[5] J. Gettys R.W. Scheifler. The x window system. *ACM Transactions on Graphics*, 5(2), 1989.

[6] Helping computers communicate. IEEE Spectrum, March 1986.

[7] Marshall T. Rose. *The Open Book.* Prentice-Hall, 1990.

[8] *P1003.0, Guide to the POSIX Open Systems Environment, Draft 15.* Institute of Electrical and Electronics Engineers, 1992.

[9] William Stallings. *ISDN: An Introduction.* Macmillan, 1989.

[10] NIST. *Proceedings, North American ISDN Users' Forum, 12, June 17 - 21.* National Institute of Standards and Technology, 1991.

[11] ISO/IEC JTC1/SC21. *Basic Reference Model of Open Distributed Processing - N2510.* 1988.

[12] NIST/ECMA. *Reference Model for Frameworks of Software Engineering Environments , NIST Special Publication 500-201 and ECMA Technical Report TR/55.* National Institute of Standards and Technology/European Computer Manufacturers Association, 1991.

[13] Object management architecture guide 1.0. Technical Report OMG Document 90.9.1, Object Management Group Inc., Framingham, MA, November 1990.

[14] The common object request broker: Architecture and specification. Technical Report OMG Document 91.12.1, Object Management Group Inc., Framingham, MA, November 1990.

[15] D. Blyth C. Boldyreff C. Ruggles N. Tetteh-Lartey. The case for formal methods in standards. *IEEE Software*, 7(5).