On the Effective Use of Software Standards in Systems Integration

D. Richard Kuhn

National Computer Systems Laboratory
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, Md. 20899

ABSTRACT

The emphasis on "open systems" in the past few years has led to the development of interface standards in almost all areas of computing: operating systems, data communications, graphics, programming languages, and others. While intelligent use of standards can solve many integration problems, the architecture of applications can significantly affect the degree of success in systems integration. This paper explains an approach to application development that helps use software standards to greatest advantage in systems integration.

1. Introduction

Software standards for programming languages have existed for many years. Database and graphics standards are more recent, and document processing language standards are newer still. The POSIX operating system interface standard 1003.1 [IEEE1988] represents the first standardization of operating system services. Additional system service standards are nearing completion. Within the next two years, it will be possible to put together a complete environment for software development and operation using products conforming to non-proprietary national and international standards.

While adherence to standards can make systems integration easier, the architecture of systems can have a significant impact on the degree to which the standards help prevent problems in integration and change. Standards solve many systems integration problems resulting from inconsistencies between major system components. This is one reason for the current interest in "open systems". Yet standards will not solve all such problems. To be effective, the standards should be compatible and must be used intelligently. This paper describes a set of national and international standards that provide an open systems environment, and defines an architecture that can help standards realize their potential for improving quality and productivity in systems development.

2. Systems Integration

For the purposes of this paper, we define systems integration as the practice of joining the functions of a set of subsystems, software or hardware, to result in a single, unified system that supports some need of an organization. We do not restrict the definition to the initial development of a system. Significant integration problems can occur during the maintenance phase of an application's lifecycle, as a result of changes either in application requirements, requiring the integration of new subsystems, or revisions of the operating system or other system services, requiring integration of the application with new system services. Fortunately, a system architecture that eases the integration of standard components during development makes modification and re-integration easier as well.

3. Industry Software Interface Standards

Responding to the demands of users for open systems, national and international standards organizations have developed standard service interfaces. A key point is that the *interface* is being standardized,

* UNIX is a trademark of AT&T

† Description of commercial products does not imply the endorsement or approval of NIST or the U.S. Government.

without regard to how the services are implemented. An example is the POSIX operating system interface: it is derived from the UNIX* operating system, but POSIX interfaces are being implemented on non-UNIX systems, such as Digital Equipment Corporation's VMS†. Before considering how to take advantage of standard services, it is worthwhile to consider some of the characteristics of software standards.

3.1. Characteristics of Standards

They represent a consensus of industry. The Institute of Electrical and Electronics Engineers (IEEE) requires 75% approval from a balloting group for a standard to be adopted. Other standards bodies follow similar rules. The development of a standard depends on the contribution of expensive personnel time and resources by industry. There is tremendous pressure to develop consensus, to ensure that the standard is approved and, above all, that it is used as a basis for products by most of the industry. A standard that is ignored by the vendors who build systems is of no value to users. The standard that results from the consensus building process often contains features that represent the "lowest common denominator" among systems provided by vendors. Other features are likely to be entirely new, the result of compromises worked out where there was no common service definition.

They change periodically. Recognizing that standards must stay reasonably current if they are to remain useful, standards bodies revise their work periodically, typically with a revision cycle of about five years. The revision period is a tradeoff between lagging the industry and changing so fast that the standard has so many versions that it is not a standard at all.

They often (but don't always) lag the state of the art. Most of the work of national and international standards bodies is involved with standardizing things that are already existing practice. The development of standards requires support from industry, which is usually not achievable unless there is a consensus already. Some recent efforts have been directed toward developing standards in relatively new ares of technology, but these efforts may not succeed if there is no common existing practice. An example is IEEE P1125, *Standard for an Object Oriented Programming Language and Environment,* whose project authorization request was withdrawn in June, 1989 [IEEE1989]. Since strong support from industry is required to approve a standard, usually only functions that are provided by most vendors are standardized. Novel features provided by one or two vendors, no matter how technically interesting, are not acceptable by the majority unless they are easy to implement. There are some exceptions, such as the OSI Class 4 Transport protocol and the POSIX Real Time extensions (1003.4). In these cases standards have been developed where there is no common existing practice, but many standards will continue to lag developments in industry.

They are typically more precise than most software specifications. Standards are developed by large groups working over several years. The focus of the effort is on making the definitions as precise as possible. However, except for data communication protocols, standards documents generally describe system functions in natural language, as do most software specifications. Despite the inherent problems of natural language specifications, standards documents tend to be more precise and complete than typical software specifications, because they are given careful review by more people than most software specifications. A standards balloting group may have more than 200 people, whose incentive for careful review is to protect the interests of their companies.

3.2. A Set of Standards for Open Systems

In its role of representing Federal agency users, the National Institute of Standards and Technology (NIST) is addressing the need for nonproprietary standards for open systems. NIST has developed a profile of such standards for use by government agencies. The Applications Portability Profile (APP) [NIST1988, Mart1989] is a family of related standards that can improve application portability and ease the integration of software developed by different organizations and based on a variety of hardware platforms. The components of the APP constitute a "toolbox" of standard elements for application system development based on non-proprietary standards. Six major functional areas are addressed: operating systems, database management, data interchange, network services, user interfaces, and programming services. The planned components of the APP are shown in Figure 1. Developers in particular application areas may wish to supplement the APP components with other relevant standards. For example, the IEEE Standard Reference Model for Computing System Tool Interconnections (IEEE P1175) [IEEE1989a] will assist developers of Computer Aided Software Engineering tools in enhancing interoperability and portability. POSIX is actually a family of standards. Its members are shown in Figure 2.

The use of APP components will allow Federal agencies to integrate systems from different vendors with greater ease than previously possible. The standards are intended to allow application portability at the source code level, so that applications can be moved from one system to another by recompiling them. No source code changes should be required in most cases. This will allow integration of products from multiple vendors into the same system, with

Function	Element	Specification
Operating System	Extended POSIX	IEEE 1003.1 (kernel), IEEE 1003.2 (shell & tools), IEEE 1003.7 (system administration)
Database Management	SQL IRDS	FIPS 127 X3.138, FIPS 156
Data Interchange - Graphics - Product Data - Document Processing	 CGM IGES & PDES SGML ODA, ODIF	 FIPS 128 NBSIR 88-3813 ISO 8879-1986, FIPS 152 ISO-DIS 8613
Network Services - Data Communications - File Management	 OSI TFA (NFS)	 FIPS 146 (GOSIP) IEEE P1003.8
User Interface - protocol, intrinsics - toolkit	 X Window System	 X V11, R3 IEEE 1201.1
Programming Services	C FORTRAN COBOL Ada Pascal	ANSI X3J11 FIPS 069-1 FIPS 021-2 FIPS 119 FIPS 109

Figure 1. Applications Portability Profile

Standard	Subject
1003.0	Open System Environment
1003.1	Operating System Kernel
1003.2	Shell and Tools
1003.3	Test Methods
1003.4	Real Time Extensions
1003.5	Ada Bindings
1003.6	Security Extensions
1003.7	System Administration
1003.8	Distributed System Services (Network File Access)
1003.9	FORTRAN Bindings
1003.10	Supercomputing Application Environment Profile
1003.11	Transaction Processing Application Environment Profile

Figure 2. POSIX Standards

significantly less concern for differences between the system services provided. Currently, software developed in one variety of C for one variety of UNIX often needs extensive modifications to reuse it and integrate it with software for a different UNIX and C. One goal of the APP is to significantly reduce such problems in the future. Integration problems cannot be eliminated entirely using standards, because most applications will still require the use of some proprietary services, and because ambiguities and imperfections in the standards may result in differences in implementations provided by different vendors. In the next section, we will look at ways to further reduce integration difficulties caused by these two problems.

4. An Architecture for the Effective Use of Standards

Given the existence of a comprehensive set of standards, we face the question of how to use the standards to the greatest advantage in systems integration. In this section we describe three models for the development of applications and consider the effect each has on system integration.

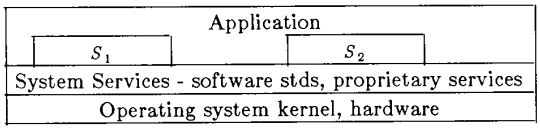
An application can be built using the following architecture:

Application
System Services - software stds, proprietary services
Operating system kernel, hardware

Using standards helps provide greater portability for the application and makes system integration easier because products of different vendors can be used interchangeably. Experience shows, though, that most applications have a few functions that cannot be provided by standard components. These will necessarily be implemented either entirely by the user, or, more likely, by functions of proprietary systems.

To a user who already owns the proprietary system, this may present little problem. For a government agency, however, choosing a proprietary system means additional justification for the purchase and considerable delay in procurement. The contract selection may be protested, so that the agency cannot obtain the proprietary system it wanted on schedule, or in some cases, cannot obtain it at all. As a result, development is slowed because some portions of the system cannot be built until the agency is certain which vendor's system services will be available to the application, and cannot be tested until the proprietary system is obtained.

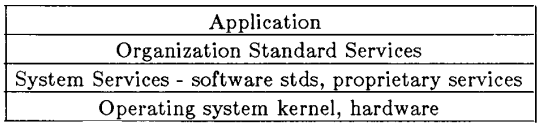
Good software engineering practice dictates that functions which are likely to change be isolated in separate modules, giving us the following architecture:



The S_i represent modules that isolate proprietary system services. These modules make integration easier by making it possible to write the application in terms of the services supplied by S_1, S_2, etc., rather than using proprietary system services directly. The advantages provided by this arrangement suggest generalizing it.

4.1. Organization Software Interface Standards

The model described above can be expanded to provide additional application independence from system services. This can be done using a layer of services that are implemented using the standard system services, giving the following model:



This service layer is a standard set of interfaces for the *organization*. The services may be simple mappings to services provided by the industry standards described previously, but it is generally better for the services to be more specific to the organization. An organization specific interface provides a means of integrating vendor-provided services with user-specific services. Applications can then be built using a "toolkit" approach, by creating unique services and integrating them with existing services [Maju1989]. This extra layer has been called a *services backplane* [McCo1989]. A similar idea is the *software backplane* described in [Brow1988]. The application calls the organization standard services, rather than calling operating system services directly. Organization standard services are functions that are needed by a variety of applications within the organization. This approach is not new. Provision of a special layer of primitives with which to develop an application is one of the traditional methods of achieving portability across a variety of operating systems and hardware. (See, for example, [Wait1977].) The remainder of this paper will examine this old idea in the context of the new software standards. It might appear that standards eliminate the need for this type of portability layer. Indeed, the service primitives provided in the past were typically at the level of operating system services, such as file I/O [Wait1977, Hans1983]. Standards now provide services at this level, but there are other reasons for organizations to build their own standard service primitives for higher-level functions.

4.2. Organization Standard Services and Industry 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. We may be building a system that needs 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. For example, Sun's NFS and AT&T's RFS are both distributed file system services, but an application that is built to use one of them cannot be easily modified to use the other.

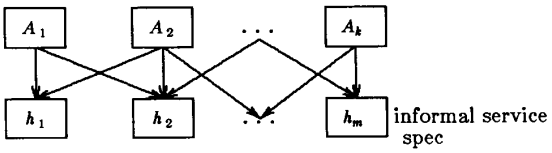
Using organization standard services isolates the application from the interface with industry standard system services. Modifications to accommodate changes in system services are restricted to the organization standard service modules. The organization standard services can be compared with objects 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 software standards thus provide independence from hardware and operating systems, while the organization standard provides independence from system services. This helps insulate the application from proprietary operating system services as well as from changes in standards.

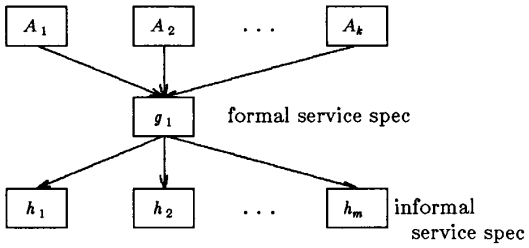
4.3. Organization Standard Services and Formal Methods

Software standards are generally written in natural language, with little or no use of mathematical formalism. This inevitably results in the kinds of problems common to informal, natural language specifications. The extensive review process removes most ambiguities and inconsistencies, but the result is usually less precise than a formal specification. IEEE Std. 1003.1 (POSIX) went through 13 revisions and was reviewed by over 200 people. Even with this extensive scrutiny, specification problems were later discovered by IEEE working group 1003.3, which defines test methods for the POSIX family of standards.

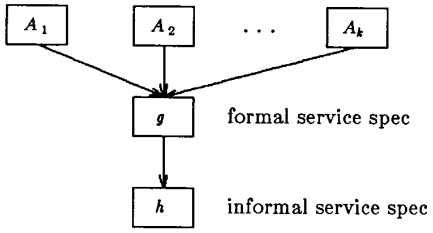
Formal methods aid in both verifying properties that cannot be shown by testing alone, such as security and safety, and in clarifying the description of system functions. In this discussion, we will concentrate on the latter use. Ideally, one would like to have system services formally specified. Sadly, there are few system service specifications that are written using anything more precise than natural language, resulting in ambiguities and inconsistencies. Writing applications to use system services requires verification of the application calls to informally specified system services, as below:



This requires km verifications against an informal specification for k uses of a function that requires m system services. The function can be created as an organization standard service that is formally specified. The applications can use the organization standard service rather than use the system services directly. Developing organization standard services that can be formally specified results in the following configuration:



This is really the same situation as traditional functional abstraction, reducing the effort to understand and verify a program by combining related functions into a single interface. But system integration problems can be reduced even when the organization standard services use only one system service, as below:



Suppose developers' understanding of the system service h is incomplete, so that some or all k uses of it are incorrect. If h is called directly by the application, and errors are discovered at system integration time, there will be up to k uses to correct. Writing a formally specified function g, that uses h, provides an unambiguously specified function that can be used in applications. The use of h by g may still be incorrect, but when the error is discovered, there will be only one place where it must be corrected. Realistically, we know that having a formal specification does not mean the elimination of *all* errors and misunderstandings, but we do expect errors to be significantly reduced [Mill1987, Hoar1984].

4.4. Organization Standard Services and Window Systems

Carrying this approach one step further, we can take advantage of the capabilities of windowing systems. Bit-mapped graphics workstations have made it possible to use direct manipulation window interfaces as the primary means of interaction with the user.

Systems such as the X Window System [Sche1986] and InterViews [Lint1989] are designed to allow separation of the user interface code from the application , resulting in the following architecture:

	User
(relatively stable)	User Interface - Organization Std
(fast change)	Application
(relatively stable)	Organization Standard Services
(slow change)	System Services - software stds
(slow change)	Operating system

In addition to separating interface and application code, the X Window System supports an object-oriented programming model, allowing developers to create customized "widgets" (screen objects). These widgets turn out to be ideal for implementing organization standard services for the user interface.

5. Experience with the Approach and Plans

Several Federal agencies have developed their own organization standard services to provide independence from proprietary products. These efforts have been successful and agency personnel have shared their experiences in NIST sponsored workshops. Some of the same agencies are now moving toward the adoption of standards such as POSIX. The organization standard services are expected to greatly simplifiy the integration of new system services as proprietary products are replaced with standards.

A more interesting development is a recently initiated project that will incorporate all aspects of the model described here into a set of formally specified standard system services. The formal specification will be done using the extended state transition language Estelle [ISO1986]. Much implementation code can be generated mechanically from the Estelle specification [NBS1987]. User interface services will be built using either InterViews [Lint1989] or TAE+ [Cent1988].

An additional topic of importance is the question of when the various open system standards should be introduced into an organization. Many organizations have existing systems that they would like to migrate to an open systems environment, but it is not often practical to change operating system interface, communications, database, and other interfaces all at the same time. Organizations will need to develop migration strategies that allow them to integrate open system standards into their operations in a system of stepwise refinements. NIST is currently developing guidance in this area [Hank1989].

6. Conclusions

Open systems standards are new to the computer industry. They can be especially beneficial in systems integration if used wisely. This paper has presented a set of standards for an open systems environment and defined an approach to use these standards to greatest advantage in systems integration. In particular, the approach helps to deal with three aspects of software standards that affect systems integration: periodic revision, missing features that result in the use of proprietary system services, and imprecise, natural language specification. The architectural approach is consistent with the "toolkit" model of systems development that has been popularized by window systems, and takes advantage of the features provided by many window systems for building user-defined components.

7. References

[Brow1988] Brown, D.W., C.D. Carson, W.A. Montgomery, P.M. Zislis, "Software Specification and Prototyping Technologies," *AT&T Technical Journal,* July/August, 1988.

[Cent1988] Century Computing, "TAE+ User's Guide", NASA Goddard Space Flight Center, Greenbelt, Md., 1988.

[Hank1989] Hankinson, A.L., "Migration to an Open System Environment, A Strategy," (draft) National Institute of Standards and Technology, November 11, 1989.

[Hans1983] Hanson D.R., "A Portable Input/Output System," *Software Practice & Experience,* Vol. 13, No. 1 (January, 1983).

[Hoar1984] Hoare, C.A.R., "Programming: Sorcery or Science?", *IEEE Software,* Vol. 1, No. 2, (April, 1984).

[IEEE1988] *IEEE Standard Portable Operating System Interface for Computer Environments,* Institute of Electrical and Electronics Engineers, Inc., New York, 1988. Piscataway, New Jersey.

[IEEE1989] *IEEE Standards Bearer*, Vol. 3, No. 3, (June, 1989), IEEE, Piscataway, New Jersey.

[IEEE1989a] *A Standard Reference Model for Computing System Tool Interconnections,* Institute of Electrical and Electronics Engineers, Inc., New York, August 30, 1989. Piscataway, New Jersey.

[ISO1986] ISO TC97/SC21, 'Estelle: A Formal Description Technique Based on an Extended State Transition Model,' ISO DP9074, 1986.

[Lint1989] Linton, M.A., J.M. Vlissides, P.R. Calder, "Composing User Interfaces with InterViews," *IEEE Computer,* Vol. 22, No. 2 (February 1989).

[Maju1989] Majurski, W., M. Ruhl, "AES Technical Architecture Evaluation," National Institute of Standards and Technology Report, Advanced Systems Division, June, 1989.

[Mart1989] Martin, R.J., "The Standards Test for Portability", *Datamation,* May 15, 1989.

[McCo1989] McCoy, W., Majurski, W., "A Structure for Developing ISDN Applications", ISDN Forum Report, National Institute of Standards and Technology, January 17, 1989.

[Mill1987] Mills, H.D., M. Dyer, R. Linger, "Cleanroom Software Engineering", *IEEE Software,* Vol. 4, No. 5, (September, 1987).

[NBS1987] National Bureau of Standards, *User's Guide for NBS Prototype Compiler for Estelle,* National Institute of Standards and Technology, Gaithersburg, Md., October, 1987.

[NIST1988] Federal Information Processing Standard 151, *POSIX: Portable Operating System Interface for Computer Environments,* National Institute of Standards and Technology, Gaithersburg, Md., September 12, 1988.

[Sche1986] Scheifler, R.W., J. Gettys, "The X Window System," *ACM Transactions on Graphics,* Vol. 5, No. 2., (April 1986).