IRM: the Integrated Reference Model for Open Systems Environments

Mieczyslaw M. Kokar
Department of Electrical and Computer Engineering
Northeastern University
360 Huntington Avenue
Boston, MA 02115

Nancy Daly
The MITRE Corporation
Bedford, MA 01730

Abstract

This paper reports results of a study of reference models for representing open system environments (infrastructures). One of the issues considered in this paper is how to compare particular reference models known in the literature. We propose a set of features of reference models on which to base such comparisons. We have found that in some cases, due to the ambiguity of definitions, it is difficult to assess whether a particular model has some features or not, and that none of the five analyzed models possesses all the features listed in this paper. As a result, we developed the Integrated Reference Model (IRM) which capitalizes on the concepts included in the previously developed models. We provide a description of the basic components of the IRM and suggest directions for refining the model if necessary.

We analyze the IRM from the point of view of the desirable features of reference models. We also outline ways in which the IRM could be used in the processes of the design of software infrastructures and in developing software using infrastructures.

1 Introduction

Software systems become increasingly large and complex. Typically, they are required to be able to run on different distributed target systems. They require such services as communications, data base management, sophisticated user interfaces, security services, and others. Additionally, they are expected to be easily adaptable to new updated requirements. For systems of this complexity traditional software development paradigms are not suitable. In order to control development risks, costs and development time the paradigms called *the spiral model* [4] and *software reuse* [20] have been proposed. The main point of the reuse paradigm is the reusability of existing components. Collections of components are kept in *libraries* (cf. [18]). When the components include operating systems, communications services, data base services, user interface services, and such, they are called *infrastructures* or *environments*. When interfaces to such environments are defined through nonproprietary specifications they are called *Open Systems Environments (OSEs)*.

Since OSEs are part of software systems, they need to be considered in both software development and software acquisition. Various OSEs need to be specified, characterized, analyzed, compared, evaluated, communicated, and learned. For all of these activities, first of all, there needs to be a language in terms of which such environments can be represented. *Reference models* are used to represent some aspects of OSEs.

Many reference models have appeared in the literature during last five years, each with its own special features, advantages and, unfortunately, limitations. Each model was developed with some special applications and features in mind, and therefore they are limited in the scope of applicability. In this paper we present an attempt to develop a reference model that covers the advantages of several existing reference models in one consistent framework and is still flexible enough to be used as a representation language for open systems environments. We call it the *Integrated Reference Model (IRM) for Open Systems Environments*.

The IRM is based upon the following existing reference models:

- National Institute of Standards and Technology (NIST) Open System Environment (OSE) Reference Model [14]
- Portable Operating System Interface for Computing Environments (POSIX) OSE Reference Model [17]
- Department of Defense (DOD) Technical Reference Model [7]
- Department of Defense Intelligence Information Systems (DODIIS) Reference Model [8]
- NIST Reference Model for Software Engineering Environments Framework [13].

In our study, we first identified the goals for developing open systems environments. These goals are listed in Section 2.1. In general, the goals are related to software development using the environments. Clearly, the existence of a representation of an OSE in terms of a reference model is a very significant, if not necessary, component, of both the software development and the software acquisition processes, that leads to the achievement of the goals of OSEs.

To rationalize the comparison of various reference models we first identified a set of *features* of open systems environments that need to be represented in a reference model; these features are related to the goals of OSEs. These features are listed in Section 2.3. Therefore, one way to compare reference models is by their ability to express those features. Additionally, since reference models are representational languages, they must possess some generic features, common to all representational languages. We present these features in Section 2.3.

In this study, we analyzed the five reference models [14, 13, 8, 17, 7] with respect to the features that a reference model should possess. Brief descriptions of these five models are given in Appendix A. Our first conclusion is that, in some cases, it is difficult to assess, without reasonable doubt, whether a particular model has some features or not. This is due to the ambiguity of definitions of reference models. Our second conclusion was that none of the five analyzed models possesses all the features listed in Section 2.3. Based on these conclusions we developed the IRM; we present the description of this model in Section 3.

Section 4 contains the analysis of the IRM with respect to the reference model features described in Section 2.3. Section 5 describes how the IRM can be used in the process to support the evaluation of open system infrastructures. Section 6 outlines the use of the IRM in the software development process. Some related research efforts are reviewed in Section 7. Section 8 presents conclusions and suggestions for future research on the IRM. And finally, Appendix B provides an example of using the IRM and discusses the elements of the User Interface services component (UI) of the IRM.

2 Goals of OSEs and Features of Reference Models

2.1 Goals of Open Systems Environments

Through this study of reference models we identified the goals for OSEs that are listed below. More details can be found in the descriptions of particular models [14, 13, 8, 17, 7].

Improve development efficiency. According to [7] the development efficiency can be improved by the following: common development (applications common to multiple mission areas developed only once), common operating environment (sharing hardware, software, and data), the use of commercial-off-the-shelf (COTS) products (reduce dependence on custom development and thus reduce time of development and ease maintenance process), and reuse of software components in custom development applications.

Enable application portability and scalability. Software developed on a particular infrastructure can be transferred from one platform to another with minimal or no modifications. Also, hardware upgrades should not affect operations. Scalability means the ability to configure applications for platforms of different size, from micros to mainframes.

Increase application flexibility. A flexible infrastructure allows for ease of incorporating changes and enhancements into the existing infrastructure at relatively low cost. This is an additional goal that was not explicitly included in the generic goals of using open system environments in the process of software development, maintenance, and acquisition [14, 13, 8, 17, 7].

Improve system interoperability. An application and infrastructure can exchange and mutually use information with other applications and systems. This may be facilitated through a common infrastructure (operating system, database management, data interchange, network services, user interfaces), standardization (common set of services to all applications), or other means.

Improve user productivity. This can be achieved through consistent user interfaces, integrated applications, and data sharing. Consistent user interfaces simplify training, facilitate development of future applications and ease of use across applications, and promote application portability. Integrated applications, such as office automation and electronic mail applications, integrated with mission specific applications, significantly reduce problem-solving time. Data sharing gives a similar advantage. Database sharing across software systems gives a similar advantage by reducing the time for database creation and maintenance. One database instead of multiples is created and maintained.

Improve user portability. With a minimum of retraining, user's skills, mental models, working strategies, and motor tendencies can successfully be transferred to different applications and systems.

Promote vendor independence. Interchangeable components, both hardware and software, supplied by different vendors can be used in the same system. One factor that promotes this goal is the availability and the use of nonproprietary specifications and open standards during infrastructure development.

Improve security. This goal is affected by such factors as uniform security policy. A uniform accreditation procedure can diminish security problems related to interoperability. Consistent security interface can result in fewer human errors.

Reduce life cycle cost. All the principles listed above support this goal which is, specifically, achieved by reducing duplication such as overlapping functionality and redundancy. This results in reduced software maintenance cost (since there is less software to maintain), and reduced training cost. Also, users moving to other organizations don't need be retrained, since newly developed software has the same user interface principles.

2.2 The Need for Reference Models

According to [14] an *Open System Environment (OSE) is an extensible framework that allows interfaces, services, protocols, and supporting data formats to be defined in terms of non-proprietary specifications that evolve through open (public) consensus-based forums. An OSE is intended to provide a set of information system building blocks with associated interfaces, services, protocols, and data formats.*

Following the spirit of the above definition, software development that is based on OSEs is an evolutionary process that begins with an OSE as a starting state and continues until the software specifications are met. An OSE thus is a significant part of the product. As such, OSEs need to be specified, designed, compared, evaluated, characterized, learned and communicated. All of these activities require a representation language, and for OSEs, reference models can play such a role. According to [14], a *reference model is a generally accepted representation of a particular application domain that allows people interested in that domain to agree on definitions and build a fundamental understanding within the scope of the model.* In other words, a reference model serves the purpose of representing the application domain and the open system environment. More specifically, a reference model should be able to represent all the elements of a computer system and the interfaces among the elements. This is especially important for Open Systems Environments, since particular elements can be developed by different vendors and a proper specification of these elements is necessary for both their correct implementation and for evaluation.

2.3 Desirable Features of Reference Models

As was mentioned earlier, we decided to compare reference models as representation languages for OSEs. First of all, starting with the goals listed in the previous section, we identified a set of features that reference models should possess in order to be able to adequately represent the important aspects of Open Systems Environments. Some of these features are generic, common to all representational languages, while others are specific to the representation of the Open Systems Environments. In the following we briefly describe the desirable features of reference models.

Interpretability. The meaning of particular representational terms should be unique and easy to interpret. For instance, it is important to clearly define the meaning of graphical primitives, such as boxes, circles, lines, and their interrelations. The topology of the graphical representation of the model should carry some clearly defined meaning. It should be stated explicitly whether adjacent horizontal/vertical lines indicate a relationship between the two services, and what kind of relationship it is. It should be clear whether proximity of two blocks means anything, etc.

Scope and flexibility. A reference model should be able to represent relationships imposed by system designers, for instance a specific reference architecture that needs to be followed due to the regulations imposed by the customer or the government []. A reference model is not a reference architecture, but it should be able to represent various reference architectures. A reference model should be flexible enough and have sufficient expressive power to describe many different designs of open system architectures. At the same time, a reference model should not be biased for or against any design solution. If a graphical representation is insufficient to represent all the relevant aspects of an OSE, formats for supporting documents should be part of a reference model.

Conciseness. A reference model should have a relatively small number of basic representational concepts. The fewer the number of concepts employed, the easier the reference model will be to understand. Also, a smaller number of the basic concepts means a more abstract representation of the system.

Representation of major system components. It should represent all major components of the OSE and of the domain. Specifically, it should include: the software platform, the hardware platform, software-to-software interfaces, software-to-hardware interfaces, protocols and supporting data formats, users of the computer system, external data stores, and communication links.

Representation of relationships among system components. It should represent all important relationships among the system components that are always present in an open system. Two relations that are normally represented in system analysis are *uses* and *part-of*. The former relation means that one of the elements needs and uses services of the other element to deliver its functionality. The latter relation means that one element is part of another element. Specifically, a reference model should fulfill the following requirements expressed in terms of these two relations.

- Any software element *uses* hardware to implement its function and therefore both software-to-software and hardware-to-hardware interfaces are needed for any software element.
- A software element possibly *uses* a number of other software elements, and this fact should be representable in a reference model.
- Users interact with the computer system through hardware (through keyboard, mouse, display) and therefore users are *part-of* hardware. Consequently, user interaction specifications are part of software-to-hardware specifications.
- External platform elements interact with the computer system through (communications) hardware and therefore communications links are *part-of* hardware. Con-

sequently, specifications of external interaction are *part-of* software-to-hardware specifications.

- External data stores are connected with the computer system through hardware and therefore external data stores are *part-of* hardware. Consequently, specifications of interaction with external data stores are *part-of* software-to-hardware specifications.

- In open systems, applications, once incorporated into an OSE, should be accessible to other software elements in the same way as other elements of the OSE. This fact should be representable in the model as a *uses* relation between the infrastructure and the application.

Software standards. It should be able to explicitly represent standards that are followed in the Software Interface Definitions (SIDs) through which particular services can be accessed within a reference model. Also, the model should facilitate the assessment of conformance to standards.

Hardware standards. It should be able to explicitly represent standards for the Hardware Interface Definitions (HIDs) through which particular services can be accessed. Also, the model should facilitate the assessment of conformance to standards.

Processing model. In a time-critical multiprocessing system, many complex control flow issues must be resolved including shared access to resources, such as memory, CPU utilization, and mass storage. Issues that arise include: deadlock avoidance and/or recovery, process starvation, race conditions, serialization, and synchronization (possibly in real time). To represent all of these issues, the overall processing model for the architecture needs to be specified. It is not necessary to specify the lower level details of the behavior of the underlying operating systems. However, it is important that the scheduling and prioritization policies (e.g., round-robin, pre-emptive scheduling) and time management be specified, since these will directly affect how the applications software architecture of multiple processes will behave.

Whether the processing model is object-oriented needs to be specified, since this will impact the way application software components should be designed. If the architecture allows processes to be specified independent of the data they operate on, application processes will need to be designed as data managers. On the other hand, if the architecture assumes that all processing is performed via concurrent objects, which package processing and data together, then the applications software components will either need to be designed that way, or will need to be encapsulated or shadowed by proxy objects which make the components appear to work that way to their clients.

COTS and Nondevelopmental Items (NDI). It should be useful to identify COTS and NDIs included in the infrastructure and assessment of the degree of integration of COTS with the infrastructure, both in terms of COTS/NDIs being able to access other services, and other services being dependent on the COTS/NDIs.

Software development tools, methods and process. It should be useful for representing software development tools, tool interfaces, tool variety (coverage), degree of integration, and conformance to standards. These tools may include software process development and management services. This representation should be unbiased, i.e., independent of any specific software development process and methods.

Security services. It should allow for the identification and representation of the services requiring special security handling. In addition, it should represent which of these services will invoke security services.

Compatibility with other reference models. Many reference models are already known in the literature, so any newly proposed reference model should be as close as possible to existing reference models. This will allow for comparison of various reference models in case a choice needs to be made.

3 Description of the IRM

3.1 IRM Physical Representation

Figure 1 is a top level view of the IRM. It shows the system elements (represented as boxes) and relations among the system elements (represented as arrows and lines).

- The hardware (HW) layer, which includes: CPUs, memory, keyboards, monitors, external data stores, communication channels, and any other hardware devices. Since the user interacts with the system through hardware, the user is also considered part of the hardware layer.

- The software (SW) layer, which includes the applications programs and all the system services (including software development tools). All software items are commonly termed *services*. Descriptions of the services are provided in the documents describing other reference models [14, 8, 17, 7].

- The interface definitions, which include protocols and supporting data formats, consist of two kinds: software interface definitions (SIDs) and hardware interface definitions (HIDs).

- The arrows represent the *uses* relation among the software elements (services). The IRM shows that these services interact through interfaces defined in the SID layer. The lines represent the *uses* relation between the software services and the hardware. The IRM shows that the software interacts with the hardware through interfaces defined in the HID layer. The *part-of* relation is represented by showing a particular service or a subsystem inside of a box.

3.2 SIDs

SIDs are either textual descriptions or formal specifications of interfaces among software components (software-to-software interfaces). They specify the *uses* relations among

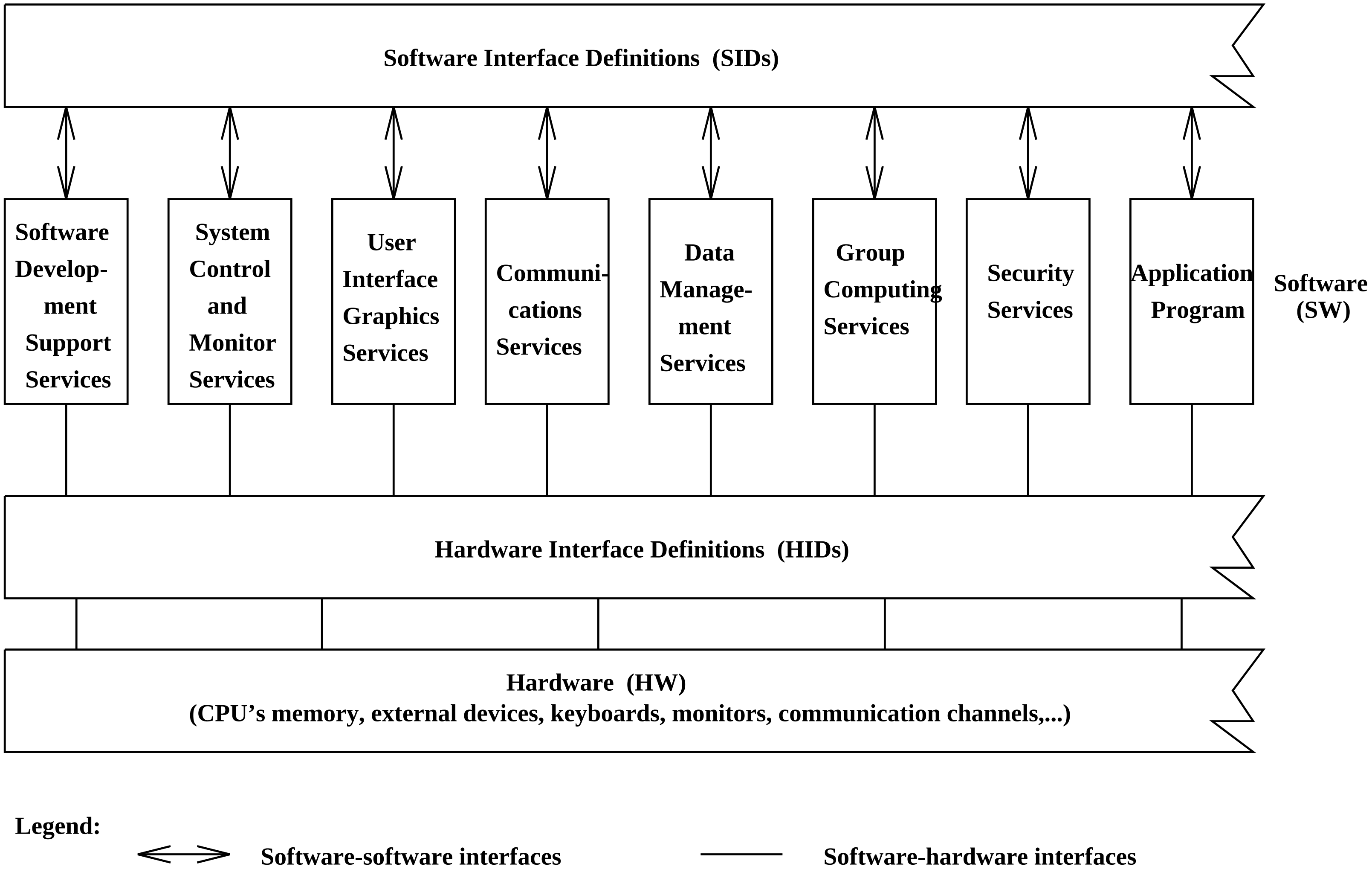

Figure 1: IRM: Integrated Reference Model

particular services. Additionally, they specify the ways in which particular services (or subservices) can be invoked by other services and/or applications, as well as what their functional and nonfunctional characteristics should be. Unlike services, these definitions do not contain any procedures or executable code.

3.3 HIDs

HIDs are either textual descriptions and/or formal specifications of interfaces between software components and the hardware platform. They specify the *uses* relations among the software services and the hardware. This includes interfaces to communication channels, CPUs, memory, external data stores and input/output (I/O) devices, including keyboards and monitors. Since users interact with the computer system through hardware, these specifications also cover user interaction. As a general rule, HID has information needed by system programmers to develop compilers, interpreters, and drivers. In particular, the HID layer contains the following definitions:

- Programming language specifications
- Event (interrupt) interfaces and latency
- Bus interfaces and latency
- Multiple CPU handling
- I/O interfaces and performance
- Data transport protocols (connectivity, interoperation, transfer)
- Time services
- Memory management services
- File services
- User interface protocols

- Command interface (getting system's attention, interrupting)
- Look and feel of character-based, block-mode, and format-based interfaces (display characteristics)
- Protocols for window interaction
- Window look-and-feel (display characteristics)
- Graphics protocols and definitions.

3.4 Supporting Documents

In addition to the graphical representation of the IRM, we propose to use, if appropriate, two additional graphical representations: an "integration matrix", showing which of the subservices are accessible to subservices in other service groups, and a layered internal structure of particular services ("hierarchy of subservices"), showing the internal hierarchical structure of some or all the services. The concept of the integration matrix is presented in Figure 2. This matrix defines the *uses* relations among the services shown as boxes in Figure 1. The general idea of the layered representation is presented in Figure 3. The purpose of this representation is to show the *uses* relation among particular subservices within a given group of services.

Services	A1	A2	B1	B2
A1		X		
A2			X	
B1	X			X
B2		X		

Figure 2: An Integration Matrix

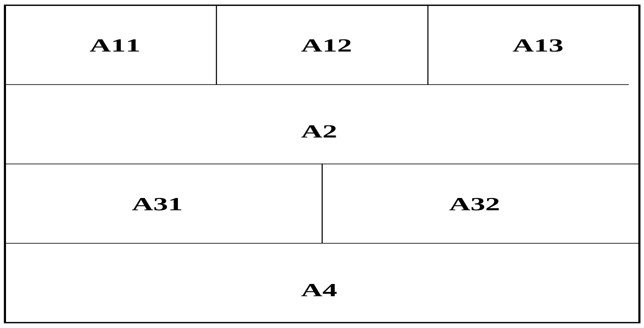

Figure 3: Hierarchy of Subservices

4 The IRM and the Desirable Features

In this section we discuss the IRM in terms of the desirable features of reference models listed in Section 2.3. We also relate the IRM to other reference models.

Interpretability and conciseness. The difficulty with the interpretability of the five analyzed models was one of the reasons we developed the new IRM reference model. We identified the following major issues:

- How to interpret the topology of the graphical representation?
- How to distinguish between internal and external interfaces?
- How to interpret "interfaces"; are they software elements or just specifications of formats and protocols?
- How to assign particular services to particular groups of interfaces, given a list of groups of services and a list of groups of interfaces?

The IRM, on the other hand, has a limited number of representational concepts (thus it is concise) with clearly defined meaning. The graphical representation of the IRM (Figure 1 shows that software runs on hardware, that interaction with the external world is carried out by means of hardware, and that there are definitions of interfaces for both software-to-software and software-to-hardware interactions. The software services (tools) represented in Figure 1 can at this moment be treated as examples. The reference model does not imply any specific connections (instances of the *uses* relation) between particular services and applications. But it allows to fix such relations at the time of defining a *reference architecture*. The fact that a particular service is next to another service does not imply that the two services interact. Instead, the *uses* relations among services for a specific reference architecture should be represented in a separate document, for instance the integration matrix (Figure 2) and the hierarchy of subservices (Figure 3).

In the IRM, the definitions of the interfaces can be clearly associated with particular system components. The programming interface definitions are associated with particular services and the hardware platform interface definitions are associated with particular hardware components. This should eliminate the vagueness present in other reference models related to the assignment of particular specifications to groups of services and hardware elements.

Scope and flexibility. Because of its simple one-layer software structure the IRM provides an unbiased and flexible representation language that can accommodate various reference architectures. Through the accompanying mechanisms for imposing constraints on the interfaces and the interconnections among services, it can support and enforce deliberate constraints imposed by the designers to comply with some standards or design philosophies.

The two kinds of representational constructs used in the IRM (the main graphical representation and supporting documents) allow for representing a reference architecture at two levels of detail. At the higher (more abstract) graphical representation level, only the blocks for particular services are shown. Their interconnections and/or accessibility can be shown in the supporting documents. Within the IRM we are able to explicitly represent the dependency of any services on any other services which is a consequence of the fact that all services are structured in one layer.

In other reference models the integration (the interservice dependencies) is not clearly represented. If we interpret the graphical representations of other models as collections of layers, where only the layer on top of a given layer *uses* that layer, then we are strongly constrained in interconnecting the layers. If on the other hand, as is stated in most of the reference models descriptions, we do not consider a reference model as a layered structure, then this implies that the feature of integration is not representable in those models.

Among the five reference models analyzed in this project, only POSIX Reference Model does not impose any layered structure. All other models do not present this kind of flexibility. The DOD TRM and the DODIIS Reference Model in particular represent the OSE in a highly structured layered way. The NIST Software Engineering Environment (SEE) Reference Model represents the infrastructure as a collection of groups of services, each of the groups represented by a block. The graphical representation indicates some relationships among the groups through the locations of the blocks. For instance, communication service is positioned at the bottom with all other services laid on top of this service; application software (tools) is connected to Object Management Services and Process Management Services, which in turn are connected to User Interface Services. Although the document describing the NIST SEE states explicitly, "It is very important to realize that the diagram must not be interpreted as a set of layers," the very fact that the diagram contains meaningful topological distinctions either inclines the reader to interpret it as a set of real interconnection constraints, or at least inhibits the nonstructured interpretation. The NIST OSE reference model has little structure – it shows security services as a layer that separates all the services from the External Platform Environment.

Representation of major system components. The elements of a reference model are software, hardware, and definitions, which include protocols and data formats. All of these elements are represented in the IRM, although the specifics of the interface definitions, i.e., protocols and data formats, are not represented as separate graphical objects. (These elements are not represented in any of the other models considered in this study.)

The IRM represents some of the components of an infrastructure that other models do not cover. In particular, the IRM explicitly represents the hardware, and makes a distinction between the hardware interfaces and the software interfaces. Explicit representation of all system elements, including the hardware, is essential because a complete system specification should include all protocols and formats for interaction with these devices. Such a need is more obvious for software that needs to be portable to other platforms, for instance for user interface and graphics services. For instance, developers of standards (cf. [11]) had to distinguish between "device independent" and "device dependent" interfaces. However, none of the reference models in our study provides means for such distinctions. The layered structure of the reviewed reference models is patterned upon the ISO Open System Interconnection (OSI) reference model [1]. However, the intent of the OSI model was to show that only the lowest layer is dependent on the (communications) hardware. The fact that each of the layers is dependent on the (other than communications) computer hardware is not addressed in the OSI model at all. In the IRM, on the other hand, this dependence is represented explicitly.

As was mentioned earlier, the IRM contains two types of interface definitions: SIDs and HIDs. Although at the first glance they look similar to the API and EEI layers in the POSIX, NIST, and DOD Technical Reference Model (TRM), the interpretation is quite different. In the other models, interfaces to external platforms were distinguished as separate entities. In the IRM, on the other hand, the distinction goes along the software/hardware lines.

For instance, the interface to the operating system services, e.g., as defined by POSIX, is defined at SID; only the interactions with the hardware (e.g., language bindings, interrupts) are defined at HID. From the application software point of view, the distinction between external platform and internal platform is not important. Whether a particular piece of data or software is residing on the same or on a different machine should be transparent to the application, and may even change with time. The application itself may also be distributed over the whole system; and, therefore, the internal/external distinction may be meaningless. For these reasons, we do not introduce such a distinction in the IRM model.

However, an explicit representation of the definition of the interface between the software and the hardware platform is essential. This is a true picture of the real configuration in which the application runs directly on the CPU, without interaction with any of the services, until a trap takes place, which causes the transfer of control to the operating system. This side of the specification is as important as the interface from the application to the environment. Especially for time-critical systems, the way in which the operating system is invoked, the priorities of the application in relation to other services, and the list of circumstances under which the context switch takes place, should be a part of a complete specification of a system. These specifications are contained in the HID layer of the IRM.

Representation of relationships among system components. The IRM unambiguously represents the *uses* and the *part-of* relations among system components. This is not the case with the other five reference models.

The IRM also represents the instances of these relations that are always present in any architecture. It explicitly shows that software *uses* hardware. This relation is important for the sake of completeness of a system specification. Since users interact with the computer system through hardware (through keyboard, mouse, display), they are *part-of* hardware. Consequently, user interaction specifications are part of software-to-hardware specifications. The User Interface (UI) definition contained in the hardware interface definition (HID) layer is a part of the realization of the concept of "uniform look and feel." This is the place to define these aspects, since the look and feel is perceived through displays, which are part of the hardware platform.

Since external platform elements interact with the computer system through (communications) hardware, communications links are *part-of* hardware. Consequently, specifications of external interaction are *part-of* software-to-hardware specifications. Similarly, since external data stores are connected with the computer system through hardware, they are *part-of* hardware, specifications of interaction with external data stores are *part-of* software-to-hardware specifications.

One of the important aspects of this reference model is that it shows applications at the same level as services (see Figure 1). This means that they can be integrated with other services, which allows us to talk about interdependencies among services and applications. Here infrastructure services can be implemented in an integrated fashion, and functions of one service can call functions of other services whenever necessary. Since some functions can simply be reused instead of re-implemented, cost of software development is minimized. Among the four reference models we studied, only the NIST SEE [13] reference model takes this aspect into account. The other three models deal with integrating an application with the infrastructure but do not cover either the integration across different applications or the integration across services within the infrastructure.

Software and hardware standards. The IRM clearly defines that standards for software-to-software interaction should be located in the SDI layer and that standards for software-to-hardware interaction should be located in the HDI layer. Other reference models do not make such a clear distinction between these two classes of standards. The IRM, similarly as the other reference models, does not impose any set of standards.

Processing model. Since an integrated software environment is an extension of the operating system, the model should support two different views of the software environment, the software view and the hardware view, and similarly the concept of the "user's view" and the "system's view" of the operating system. This distinction is important for the understanding of the processing model – how does software interact with the hardware. It was stated earlier that the IRM model allows for definition of both software-software and software-hardware interfaces and thus addresses this requirement. In the other reference models this issue is not clearly defined. Similarly as the other reference models, the IRM does not provide means for distinguishing object oriented systems from procedure based systems.

COTS and Nondevelopmental Items (NDI). As it was stated earlier, the IRM shows applications at the same level as services (see Figure 1). This means that COTS and NDIs can be integrated with other services. Among the reference models we studied, only the NIST SEE [13] reference model allocates slots for integrating additional software elements.

Software development tools, methods and process. Both software being developed and software to support the software development process and management (CASE tools) can be explicitly represented in the IRM model. Therefore, this model can be used to discuss the software engineering-related issues. The top level representation of the IRM can be used as a top level representation of either a reference architecture or a specific architectural design. At this level specific groups of services and subservices, as well as elements of computer hardware, can be shown. More architectural information can be added to this representation by showing how particular services are structured into layers and/or parallel blocks. Interfaces and/or standards can be defined in the SDI and HDI layers.

The IRM can also be used for discussing the software development process. For instance, in the development phase of an application, the application is not the program that is being executed. During the development time the developer executes some software development services such as editors, compilers, debuggers, linkers, simulators, testers, etc., as well as some software engineering process services like software metrics calculation, cost and effort estimation, project tracking, etc. Only when development is finished will the application become the final product that runs on the hardware platform and calls on other services. This situation may be reversed again should a need for further enhancements arise.

Other reference models include software development tools and services, but the use of these models for analyzing software engineering issues suffers from all the problems discussed above.

Security services. The IRM allows for the identification and representation of the services requiring special security handling (see Figure 1, but it does not enforce the use of security services by all other services. Through the use of the integration matrix and/or the subservice hierarchy (Figures 2 and 3) the designer of a reference architecture can impose such a requirement selectively.

The NIST reference model [14] shows security services at the bottom of the layered hierarchy, which suggests (although it is not clearly defined) that all other services must use security services. The TRM model [7] shows security services as one half of the bottom layer, with the other half occupied by the system management services. It is not clear how to interpret such a representation. Some other reference models do not mention security services.

Compatibility with other reference models. The IRM was developed as a result of an analysis of five reference models [14, 13, 8, 17, 7]. We have incorporated as many features of these five models as possible into our model.

5 Using the IRM for Evaluating Open System Infrastructures

One of our objectives is to use the IRM as a mechanism to evaluate and compare infrastructures during the source selection process and during the infrastructure development process while on contract. For source selection infrastructure evaluations, the offeror would be required to complete the IRM, which describes their infrastructure components and architecture, in order to allow to understand the infrastructure to verify that it satisfies the objectives of the system development. One of the most important reasons to use the IRM is that it provides a uniform and unbiased representation of the offeror's or contractor's infrastructure. In the following section we present the description of the offeror s or contractor's part in this process. Then we briefly describe examples of features that can be analyzed using the developed IRM-based representation of the infrastructure.

5.1 Developing a Representation of the Infrastructure in the IRM framework

The first activity in the infrastructure evaluation process is to represent the infrastructure in terms of the IRM. This activity consists of the following steps:

Develop the pictorial representation of the infrastructure. The offeror or contractor shall develop a pictorial representation of the infrastructure that is being considered during the evaluation. This representation shall be done in a fashion similar to the IRM (Figure 1). Included in this picture should be all the major groups of services that constitute the infrastructure.

Provide a textual description of the infrastructure. The textual description should include a discussion of which of the general goals and features of open systems environments are included in the offeror's or contractor's infrastructure (see Sections 2.1 and 2.3.) The goals and features of the infrastructure should be discussed in the context of the major groups of services and subservices within these groups.

Develop an integration matrix. Using Figure 2 as the sample format show, which of the subservices in particular groups are accessible to other services or subservices from other groups, and which of the services or subservices from other groups are allowed to access particular subservices. For instance, referring to Figure 2, service A1 can call only service A2 within the same group, A2 can call B1, B1 can call A1 and B2, and B2 can call only A2.

Show the hierarchy (layers) within each of the groups of services. Using Figure 3 as the sample format, this representation should show the hierarchical calling dependency within particular groups of services.

Provide the interface specifications for each of the services. The interface specifications relate to the software interface definitions (SIDs) shown in Figure 1. The interface specifications should be provided on a per service basis and should include and distinguish between inter-service interfaces and interfaces available to the application software. The interface specification should identify the specific standards followed, if applicable.

Provide the list of standards with which hardware interfaces comply. The interface specifications relate to the hardware interface definitions (HIDs) shown in Figure 1. The hardware interface specifications should be provided on both a per service basis (including application software as one of the services) and on a per group basis, as listed in Section 3.3. The interface specification would identify the specific standards followed, if applicable.

In order to gain a better understanding of the issues involved into developing such a representation we used an abbreviated example of the user interface service and developed its representation using the format of the IRM and the instructions listed above. This exercise is described in [5]. It exemplifies the level of detail that an offeror or contractor should include in the response to the infrastructure evaluation using the IRM.

5.2 Evaluating the Infrastructure

Once an IRM-based representation of the open system infrastructure exists, it can be analyzed from various points of view, depending on the importance and priorities associated with particular features. Ideally, this evaluation should be performed by a team of experts from both the offeror (contractor) and the requestor (sponsor). Below we present an example of features that can be analyzed using this representation.

Consistency of SIDs. Using the integration matrix (Figure 2), identify all possible interactions among services. Check the consistency of all of the software interfaces.

Coverage. Check if all of the generic services and application-specific services (needed for the particular application) are included in the proposed infrastructure and represented in the IRM.

Degree of integration. Using the integration matrix (Figure 2) representing dependencies of particular services on other services, assess the degree of this interdependency.

Conformance to standards of the SIDs. Using both the hierarchical representation (Figure 3) and the integration matrix (Figure 2), identify all the software interface definitions (SIDs) through which particular services need to be accessed. Using the textual description provided by the offeror, identify standards used in their specification. Analyze the conformance of the interfaces to these standards.

Conformance to standards of the HIDs. Using the IRM representation (Figure 1), identify the hardware interfaces. Using the textual description provided by the offeror, identify the standards used in their specification. Analyze the conformance of the interfaces to these standards.

Variety and quality of software development tools. Among services represented in the IRM (Figure 1), distinguish software development tools. Analyze the functionality of tools (variety) and the degree of the integration.

Compatibility of COTS. Analyze SIDs and HIDs of the COTS that will be integrated into the infrastructure. Analyze the compatibility of the COTS with the infrastructure.

Security services. Identify which of the services require special security handling. Analyze the security services provided by the infrastructure. Analyze the integration matrix and check if all of those services invoke security services.

Flexibility and adaptability of the infrastructure. Consider a number of scenarios of change in the requirements and functionality of the application. Assess what and how much would need to be changed in either the application or the infrastructure or both to accommodate the required changes.

6 Using the IRM for Developing Open System Infrastructures

6.1 Software Infrastructure Design Characteristics

In this section we present some of our suggestions on the design of infrastructures. The infrastructure services can consist of COTS, Government-off-the-Shelf (GOTS), and/or contractor-developed software. These services need to be integrated to provide the distributed processing required by the mission application. The infrastructure services and development environment/methodology should be designed to provide modular, localized interfaces which, in turn, provide the basis for flexibility, extensibility, and evolvability of the infrastructure services and the mission application software.

The software infrastructure shall have the following design characteristics:

- It shall use COTS to the maximum extent possible. It shall capitalize on industry accepted standards and commercially available products to the maximum extent possible to support the mission requirements.
- It shall use well-defined application program interfaces (APIs) between the mission application software and software infrastructure services.
- It shall be portable and have the ability to execute across heterogeneous platforms (POSIX compliance).
- It shall demonstrate extensibility and evolvability of the services to support emerging standards and commercially available products.
- It shall demonstrate that the infrastructure is sufficiently flexible and has the capability to support changes and extensions to the mission application software.
- It shall not preclude the use of specific programming languages, for instance Ada, to implement mission applications.

- Interdependencies between the infrastructure services shall be optimized so that the impact of change is localized.
- Use of an existing software infrastructure shall enhance the maintainability of the system.
- A software methodology/environment shall exist to support the development of the mission application software in the context of the infrastructure services and environment.
- Methodology for software infrastructure and development should reduce risk and life-cycle costs.
- Issues related to performance, including performance monitoring and performance optimization, should be addressed.

6.2 Developing Software Based on OSEs

The problem that the software developer is facing is how to develop applications using open system infrastructures. Anybody who had to develop software that incorporates X Windows or Windows had to overcome such a problem. First of all, one needs to understand the functionality, the structure, the scope and the limitations of such an environment. One needs to develop an internal model of the whole environment. The documentation that comes with such environments is either too general or too specific. The documentation usually gives a very general description of the whole environment, relatively good description of the functionality of particular modules, and the source code. A reference model, like the one presented in this paper, provides the structure that serves as a glue for all of the pieces necessary for the understanding of all of the aspects of the environment. Only such an in-depth understanding allows one to begin to develop software for a particular environment.

The development of a system should capitalize on an existing software infrastructure which portrays the design characteristics discussed in Section 6.1. Leveraging on an existing infrastructure will reduce the initial development time and let the contractor focus on developing the mission-critical software rather than the support services.

An existing software infrastructure that possesses a stable software development environment and the necessary support services will lend itself to supporting an incremental build and integration system development strategy. An existing infrastructure will also allow the contractor to use the actual software to develop rapid prototypes and demonstrate the system performance as it is incrementally developed.

While the software infrastructure offers many benefits, significant challenges exist. Two of the most significant challenges not addressed via the software infrastructure approach are:

- Maintaining interoperability between different infrastructures
- Ensuring that the selected infrastructure does not preclude the use of future standards and technologies.

The software infrastructure approach discussed in this paper leaves the interoperability between infrastructures to be addressed and resolved by the application software, except as facilitated by common communication standards and/or infrastructure services.

It may not be possible to ensure that the infrastructure does not preclude the use of future standards. However, there are a few things that can be done to mitigate this risk.

- Evaluate each infrastructure on a case-by-case basis for the software dependencies on standards and technologies.
- Evaluate the APIs for their ability to support extensibility and flexibility of the infrastructure services.
- Perform a series of flexibility exercises while the system is under development, to ensure that the design characteristics are being considered during the development process.

- Evaluate the risk of using the selected infrastructure for a prespecified minimum duration life cycle.

7 Related Research

In the literature search related to this study we had not found any paper that would address the issue of comparing various reference models. The closest result was Holmes' paper [10] on the use of reference models in the process of standardization of computer systems. In that paper, various methods for developing reference models are proposed. For instance, the Interface Oriented Functional Decomposition Modeling Technique stresses interfaces among software components, while the Resource Based Model represents standards related to four hardware components (processor, storage, communications and display) arranged into abstraction layers, with the application positioned on top of these layers. Among the models that we investigated, the NIST model [14] is the closest to this representation. However, while this representation gives a way for arranging standards related to particular hardware components, it does not provide any means for combining these standards with software interface standards. It is important to notice, however, that [10] recognizes the need for representing both hardware and software related standards, although not in one representational model.

There exist papers on the application of standards developed using reference models. For instance, [11] describes the application of standards in the process of testing computer graphics. In this kind of work authors are mainly concerned with their application domain and not with the particular model in which the standards are represented.

Recently, much research is being done in the area of software architectures. Since reference models are the means for establishing reference architectures, this area is related to the problems discussed in this paper. The DARPA sponsored program of Domain Specific Software Architecture (DSSA) contains several projects that deal with software development similar to the one described in this paper, i.e., a reference architecture for a specific domain is developed first and then the architecture is instantiated with pre-existing components, components are composed, adapted and verified [18]. In some cases, special purpose languages are developed to describe software architectures, e.g., LILEANNA [21] or MetaH [22]. Recently, an entire book was devoted to software architectures [19], in which various styles of organizing software systems are described. While each of the styles in [19] is represented in a different notation, the work of Gerken [9] uses category theory to formally specify software architectures in one common framework. The great advantage of Gerken's approach is that once an architecture is specified, a particular implementation can be tested against the specification in order to verify that it adheres to the specified architecture.

All of these developments in software architectures have a very clear and direct relation to the reference model presented in this paper. In order to clearly understand the relationship we need to realize that the IRM is intended to represent the structure of a whole computer system, i.e., hardware, software, users, peripherals, and all of the services that a specific Open System Environment provides. The models used in software architectures, on the other hand, deal primarily with the structure of one element, i.e., software. They are based on the presumption that software-to-hardware interactions are guaranteed to adhere to some standards, and that this is the responsibility of the particular operating system. In this paper we indicated some directions of refining the IRM in order to be able to represent more details of an OSE, for instance an integration matrix or a hierarchy of services. These additional representations can be either further refined, or replaced, by the models used in describing software architectures, e.g., by an architecture specification language or by the category theory specifications. Nevertheless, when developing a computer system, one needs to deal with both system specification and software specification. A representation of the whole system, like the one proposed in this paper, is necessary in order to facilitate proper relation between system engineering and software engineering.

8 Conclusions

This paper reports results of a study of reference models for representing open system environments. One of the main issues in this study was the decision on how to compare particular reference models known in the literature. Our conclusion was that it can be based on goals of open system environments and features of reference models. Through our study, we first identified a set of goals for open system environments. Secondly, we proposed to consider reference models as representational structures, and identified features of reference models that are appropriate for representing open system environments in a reference model. As the next step, we studied several reference models. The initial objective of the reference model investigation was not to develop a new one, but to select a reference model that satisfied our view of the C^2 processing domain and open systems. However, upon review of the reference models, we have found that (1) in some cases, due to the ambiguity of definitions of reference models, it is difficult to assess, without reasonable doubt, whether a particular model has some features or not, and (2) that none of the five analyzed models possesses all the features that are important to the development of a C^2 system. As a result, we developed the IRM which capitalizes on the concepts included in the previously developed models and includes the benefits of open systems provided in the models. Some of the advantages of the IRM are listed below.

- It does not impose any model-related biases on the system design and representation, but it allows one to impose constraints, if necessary.
- It explicitly spells out the meaning of all representational elements and, thus, it is clear how to interpret open system environments represented in the IRM.
- It allows us to express both the hardware and the software constraints that need to be imposed on the interfaces.
- It allows us to show not only how a given application accesses particular services, but also how services can access the application and other services.

- We can include software process management services in the definitions of reference models in exactly the same way as other services.

One of our conclusions in this study was that we can more effectively communicate the goals of open systems and the software infrastructure services required to satisfy a specific system via a tailored reference model (IRM). Also, we found that reference models can be used to gain insight into particular infrastructure designs and implementations during the acquisition process. Therefore, we suggest that reference models be used in a different manner than they are often presently being used (i.e., a vehicle in which to identify and levy standards on system development contractors). In this paper, we recommend that the software requestor should provide the IRM (or a tailored version of another reference model) to an offeror with the Request for Proposal (RFP) as a way of describing in detail the requirements and characteristics of the software infrastructure required to implement the system being acquired. We also recommend that the offerors describe their infrastructure in the context of the reference model. This will allow the requestor to gain an understanding of an offeror's software infrastructure design and implementation in a common and more structured format.

It was not our intention to imply that the reference models analyzed as part of developing this paper are not adequate or that other existing reference models are not adequate for describing the infrastructure services. However, it is strongly recommended that the service descriptions provided in the reference models be all inclusive of the capabilities and attributes required in the target system. Further, the service description in the selected reference model may require tailoring to accommodate the requirements of the target system under development.

Acknowledgements

Ron Couture was the primary author of the User Interface Services presented in Appendix B. The authors wish to acknowledge the support and guidance provided by Steve Crisp.

The first author thanks Joseph DeRosa for his support of the author's system engineering efforts, in particular of sensor/data fusion systems.

References

[1] Information processing systems - open systems interconnection - basic reference model. Technical Report IS 7498, ISO, 1984.

[2] Global protection against limited strikes (gpals) human-computer interface (hci) style guide. Technical report, DoD, 1992.

[3] Ieee recommended practice for graphical user interface drivability. Technical Report P1201.2 Balloting Draft 1, The Institute of Electrical and Electronics Engineers, New York, 1992.

[4] B. Boehm. A spiral model for software development and enhancement. *Computer*, 21, no. 5:61–72, 1988.

[5] R. G. Couture, N. Daly, and M. M. Kokar. The command and control (c2) infrastructure reference model (irm). Technical report, MITRE, Bedford, MA, 1994.

[6] DISA. Dod human computer interface style guide, version 1.0. Technical report, Defense Information Systems Agency, Center for Information Management, Washington, D.C., 1992.

[7] DISA. Technical reference model for information management, version 1.3. Technical report, Defense Information Systems Agency, Center for Information Management, Washington, D.C., 1992.

[8] DODIIS. Dodiis reference model for the 1990's. Technical report, Department of Defense Intelligence Information Systems, Arlington, VA, 1991.

[9] J. Gerken, Mark. *Specification and Design Theories for Software Architectures.* PhD thesis, Air Force Institute of Technology, 1995.

[10] E. Holmes, J. A development of reference models for computer systems. In *Proceedings – Computer Standards Conference*, pages 65–75, 1988.

[11] B. Kirsch, C. Pfluger, and C. Egelhaaf. Conformance testing for computer graphics standards. *Computer Standards and Interfaces*, 12:35–42, 1991.

[12] NIST. The user interface component of the applications portability profile. Technical Report FIPS 158, National Institute of Standards, Gaithersburg, MD, 1990.

[13] NIST. Reference model for frameworks of software engineering environments. Technical Report 500-201, National Institute of Standards, Gaithersburg, MD, 1991.

[14] NIST. Application portability profile (app) – the u.s. government's open system environment profile, ose/1 version 2.0. *Computer Standards and Interfaces*, 15:539–591, 1993.

[15] A. Nye. *Xlib Programming Manual.* O'Reilly & Associates, CA, 1990.

[16] T. O'Reilly. *X Toolkit Intrinsics Reference Manual.* O'Reilly & Associates, CA, 1990.

[17] POSIX. Standards project: Draft guide to the posix open system environment. Technical Report P1003.0/D15, IEEE, New York, 1992.

[18] R. Prieto-Diaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34, No.5:88–97, 1991.

[19] M. Shaw and D. Garland. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[20] W. Tracz. *Software Reuse: Emerging Technology.* IEEE Computer Society Press, 1988.

[21] W. Tracz. *Formal Specification of Parameterized Programs in LILEANNA.* PhD thesis, Stanford University, 1993.

[22] S. Vestal. Software programmer's manual for the honewell aerospace compiled kernel (metah language reference manual). Technical report, Honewell Systems and Research Center, 1993.

Appendix A: Overview of Reference Models

In this appendix we provide a brief overview of five reference models. More details can be found in the documents describing particular models [14, 13, 8, 17, 7]. For each of the reference models, we provide a short discussion in which we identify some of the shortcomings of the models and the accompanying documents.

NIST's Open System Environment (OSE) Reference Model

The NIST OSE Reference Model (see Figure 4) consists of two kinds of elements: *entities* and *interfaces*. The model entities are: Application Software, Application Platform (both software and hardware), and External Environment.

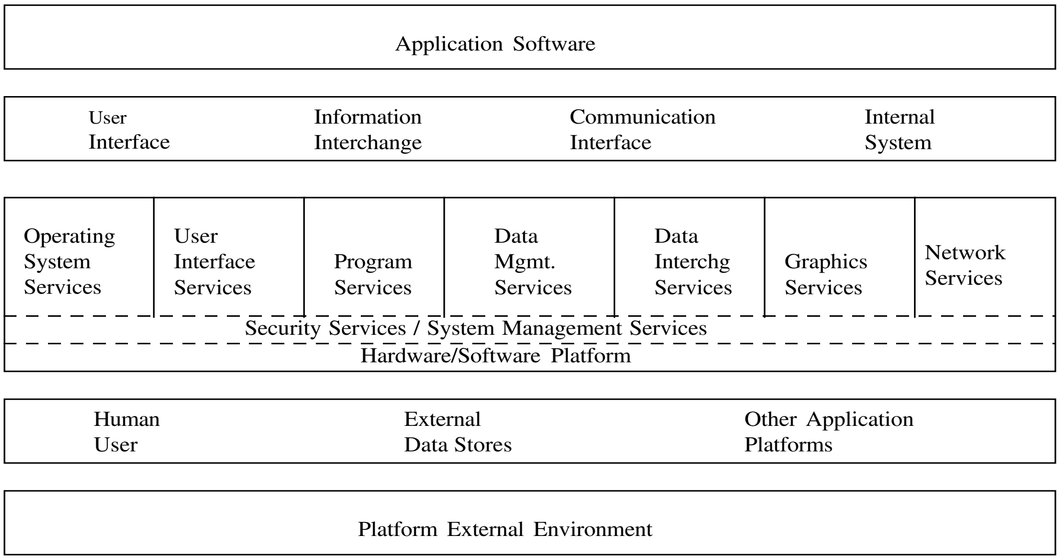

Figure 4: NIST OSE Reference Model

The Application Software is the software that is specific to a customer's application that supports particular business needs. This element includes data, documentation, training and programs. The Application Platform is a collection of hardware and software components that provide services used by the application programs. The Application Platform entity consists of seven groups of services, as shown in Figure 4. The platform External Environment entity consists of system elements that are connected to the application platform, including Human Users, External Data Stores and systems executing on Other Application Platforms.

There are two kinds of interfaces: *application program interfaces (API)* and *external environment interfaces (EEI)*. According to [14], the API is a set of functions between the Application Software and the Application Platform. They are grouped into four groups: User Interface, Information Interchange interface, Communication Interface and Internal System interface. The EEI consists of three *transfer services* from/to: Human User, External Data Stores and Other Application Platforms.

NIST OSE RM: Discussion

Some of the definitions provided in [14] are not clear. For instance, the definitions of reference model and OSE reference model describe the reasons for having a reference model rather than what a reference model is. The definition of the interfaces are ambiguous. In [14], they are defined as "functions", which suggests that they are executable software. Other documents that use the NIST model interpret them as "specifications". Additionally, it is not clear what is the relationship between the APIs and the application platform services. Also, the NIST OSE model does not explicitly represent hardware. Instead, hardware is treated simply as part of the application platform. The consequence of this is that not only one of the important computer system elements is not explicitly represented, but also the relations between hardware and software (software-to-hardware interfaces) are not representable. The application in the NIST model is represented as an element that uses other services. The model cannot represent the fact that the application can also be used by other applications and/or services. The description of the model [14] does not provide information on the external environment interfaces (EEI).

The POSIX Open System Environment Reference Model

The POSIX Reference Model has been developed to describe, among others, the POSIX Open System Environment (OSE), and in particular to compare and classify standards. The POSIX OSE is a collection of concepts that provide for user requirements and standards specification. The POSIX OSE, in addition to the reference model, includes service definitions, standards and profiles [17].

The top level view of the POSIX OSE Reference Model is represented in Figure 5. Similarly as the NIST model, it consists of three *entities* (Application Software, Application Platform, and Platform External Entity) and two *interfaces* (Application Program Interface (API) and External Environment Interface (EEI)).

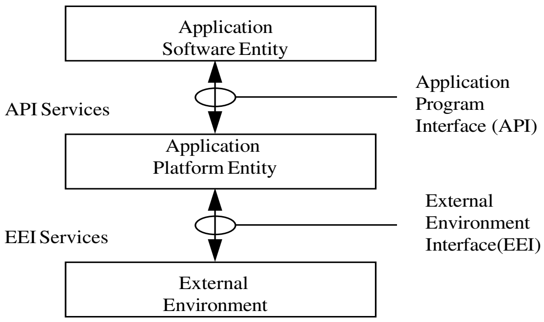

Figure 5: POSIX OSE Reference Model (outline view)

The Application Software entity is further decomposed into *elements*: Program Element (source code, command/script files), Data Element (user data, application parameters, screen definitions), and Documentation Element (on-line documentation only).

Although the Application Platform includes both hardware and software, the finer view of this entity shows *services*, which represent software. Apparently, the intent here is to make the configuration related details of the platform transparent to the application software. There are four groups of services: System Services (Language Services, System Services), Communication Services, Information Services (Data Services, Data Interchange Services, Transaction Processing Services), Human Computer Interaction Services (User Command Interface Services, Character-Based User Interface Services, Windowing System Services, Graphics Services, Application Software Development Services). For each of the services a more specific reference model is provided in [17].

The External Environment includes (see Figure 6) People, Information Interchange Entities (removable disk packs, floppy disks, security badges), and Communication Entities (phone lines, local area networks, packet switching equipment).

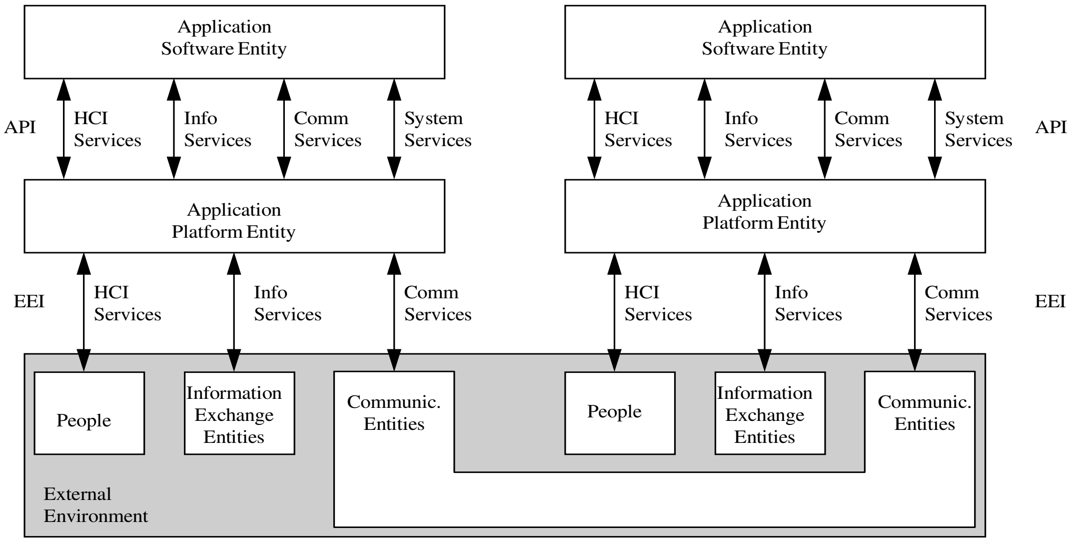

Figure 6: POSIX OSE Reference Model

The API is represented by arrows. The API is subdivided in four groups according to the partition of services. Therefore the interfaces are associated with particular services as described above. Only the System Services API is subdivided further into programming language API (language primitives), language-independent service specifications (platform's services such as interprocess communication, inter-object messages, access to user interface and to data storage), and language binding API (services for translating language-independent specifications into language-specific specifications).

The EEI consists of three groups: Human-Computer Interaction Services (standards related to CRT displays, keyboards, mice, audio input/output), Information Services (format and syntax for interacting with data storage devices), and Communication Services (protocol, syntax and format for communication).

POSIX OSE Model: Discussion

Although in the POSIX OSE model the explicit association of the API with particular services provides a clear interpretation of particular groups within the API, the specification of the EEI is still not precise. Especially, the separation between the API and the EEI (see the desired feature of *interpretability* in Section 2.3) and the impact of these two specifications on the goals of open system environments, i.e., portability and interoperability, are not clear. For instance, although the primary EEI's responsibility is to provide interoperability, it also supports user portability. Similarly, user and data portability are provided by both types of interfaces. E.g., for windows, the program interface standards for accessing windows are associated with the API, while the user standards for windows are associated with the EEI. Requests for services issued through the API are internally translated into requests at EEI.

Additionally, the POSIX OSE has the same drawbacks as the NIST model due to the fact that it does not explicitly represent hardware and that the application is represented at a higher level (on top of) than the application platform (see Section 8).

DoD/DISA Technical Reference Model (TRM)

The DoD TRM reference model [7] is a combination of the NIST model [14] and the POSIX model [17]. We do not provide a separate figure for the TRM, since the graphical representation of the TRM is the same as the POSIX representation: three entities and two interfaces. The entities are represented as boxes, while the interfaces are represented as arrows. But the components of the Application Platform entity – the services – include, in addition to the groups of services specified in the NIST model [14], the Kernel Operations, Commands and Utilities service. The six groups of services are further decomposed into a number of subservices. This decomposition was helpful in matching standards for particular services. Standardization was the main goal for developing the TRM.

TRM: Discussion

Since the TRM is a combination of the POSIX and NIST models, all of the discussion of the two previous sections apply to this model. One feature that is specific to the TRM is that since it is a refinement of the other two models, with more detailed information about subservices, it includes a graphical representation of the subservices. The DoD document [7] states that the representation does not imply a layered structure. Yet for some reason, some of the service groups have a special place within the box representing the application platform. For instance, Security Services and System Management Services occupy the bottom layer of the box, with Kernel Operations Commands and Utilities, Operating System Service and System Management and Security located on top of them. The remaining six groups of services are on top of the previous three. The more complex topology of the application platform does not seem to be justifiable. Either it should be represented simpler, like in the NIST model, or the topology should carry a meaning, in which case the layout would have to be redesigned.

DODIIS Reference Model

The *generic* DODIIS reference model consists of eleven major components, as shown in Figure 7. The services in this model are classified into five groups; the classification is different from any other reference model. Although three of these groups use the term "object", it is not meant to be an object as in object-oriented software development. Objects in this model refer to "data objects". Each of the services is defined further by a *layered* set of functions in the *detailed model* (not shown in this paper). The Operating System Services are shown as a layer underneath the five groups. Although the model shows the Hardware Platforms, the document [8] indicates that the goal is to have software that is independent from the hardware platform.

Since the main focus of the DODIIS model is information, it includes a separate entity, Stored Information, in which the information architecture is represented. System Management and Security services play a special role; they are represented as a base

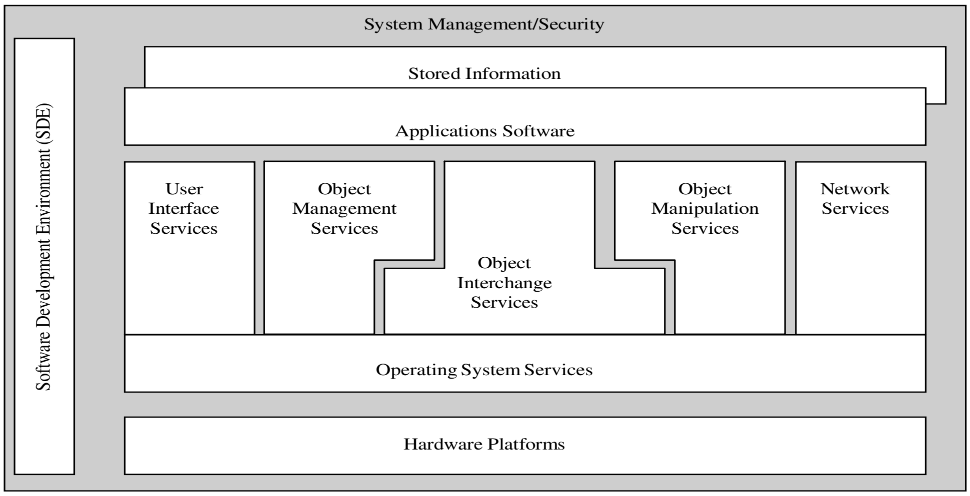

Figure 7: DODIIS Reference Model

for the rest of the system. The DODIIS document [8] lists five system management functions: Configuration Management, Fault Management, Performance Management, Accounting Management and Security Management. This document states that all five system management functions should be included in each of the service areas.

DODIIS Reference Model: Discussion

The main difference between the DODIIS model and all other models is that it does not explicitly represent interfaces, even though API and EEI are mentioned in [8]. Consequently, even though it represents some of the system elements, including hardware and data, it still is ambiguous about representing relations among these elements. This ambiguity can be resolved by selecting specific standards for the functionality contained within each rectangle in the model representation. The semantics of the graphical representation is not clear. Although the DODIIS document [8] states that the five services are composed of layers, it is not clear whether the position of a rectangle in the model above another rectangle always means the hierarchy in the system, or not. Many questions are difficult to answer. For instance, does each of the five service groups have interfaces to the operating system? How is Applications Software related to the five groups of services? Can Application Software have interfaces to the operating system? How is Stored Information related to other components? How is the Software Development Environment related to the other components, like Operating System Services and Applications Software? These and other questions are related to the desired features of reference models: interpretability, scope and flexibility, and representation of system components.

NIST's Reference Model for Software Engineering Environments (SEE) Frameworks

The main aim of the SEE reference model [13] is to provide means for describing and comparing software engineering environments, i.e., of systems which provide automated support of the engineering of software systems and the management of the software process. Therefore the goal of the SEE model is somewhat different than the goals of the other models discussed in this paper. The main focus of this model is integration.

The SEE model groups functional capabilities of software engineering frameworks into groups of services. It includes the five groups shown in Figure 8: Object Management Services, Process Management Services, User Interface Services, Communication Service, Tool Slots (place holders for extending a framework), and two groups of services that are not included in the figure: Policy Enforcement Services, and Framework Administration and Configuration Services. Each of the groups is further decomposed into subservices and extensive description is provided for each of the subservices in [13].

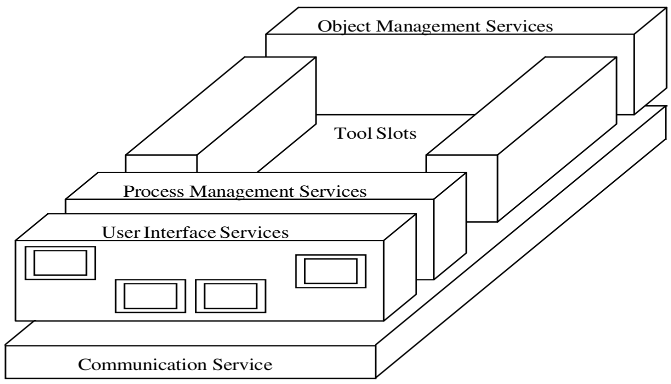

Figure 8: Software Engineering Environments (SEE) Frameworks Reference Model

The SEE reference model requires that each of the subservices be described in a structured way using the following eight "dimensions": *conceptual* (only functionality described at a conceptual level, no implementation detail), *operations* (list of operations and their functionality), rules (pre and post conditions and constraints), types (data model and meta-data information), *external* (how a particular service can be made available to other services), *internal* (implementation issues), *related services* (which services interact and how), and *examples*. The NIST document [13] includes descriptions of all the basic services in all these dimensions.

SEE Reference Model: Discussion

The goal of the SEE reference model can be considered as a subgoal of the other reference models, since all of them include software development tools. The SEE model, however, is much more specific in describing software engineering environments than any of the other models.

The information content of the graphical representation of this model is rather limited. The NIST document [13] states that the grouping of the services is just for the convenience of representation and for the purpose of discussing various aspects of software engineering frameworks. The graphical representation should not be interpreted as a set of layers. Consequently, from the point of view of desired features of reference model, it is limited in the areas of scope and flexibility and in the ability to represent major computer system components and relations among the components. However, it should be noted that the associated description in terms of the eight model dimensions provide much more information than other models.

Appendix B: User Interface Services in the IRM

This appendix provides an example of using the IRM and discusses the elements of the User Interface services component (UI) of the IRM (more detail can be found in [5]). These services collectively support the goal of Open System Environments as stated in Section 2.1. We show that both the interfaces to and among the UI services can be represented clearly and precisely using the IRM. This goal would be difficult to achieve using any one of the other reference models.

Overview of the UI Services

The top level of the User Interface services of the IRM are represented in Figure 9 using the IRM representation itself. There are five different kinds of UI services (boxes within the shaded region in the figure). They are described below.

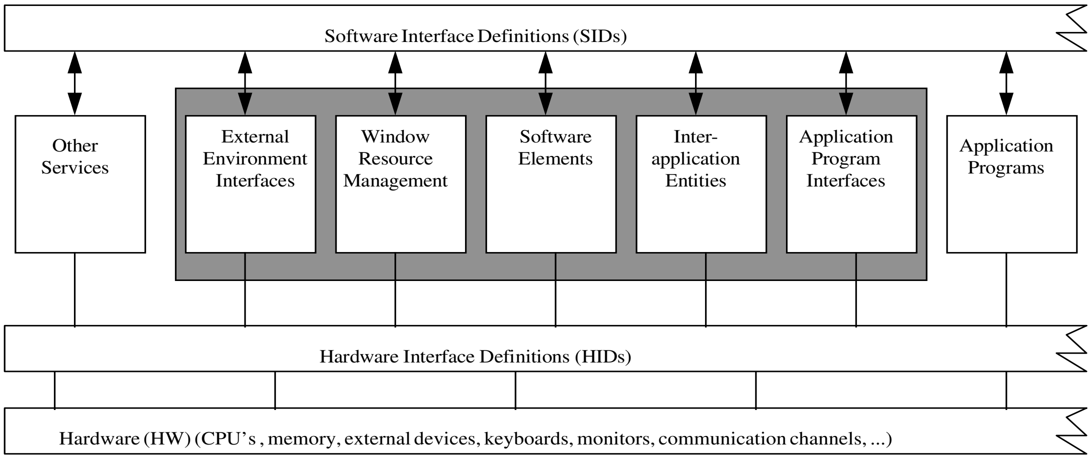

Figure 9: User Interface (UI) Services

Application Programming Interfaces (API) are services available to application programs (clients) for interacting with a graphical windowing system (the X server) which, on the other end, supports interaction with the users. Since these services can be called by the application programs, the formats and protocols that are used by the application programs for accessing these services are included in the SID layer of the IRM. We provide more details on these services in the next section.

External Environment Interfaces (EEI) provide interfaces to other IRM services and other entities external to the UI, such as the user. The definitions of these interfaces within the SID layer cover the interaction with all other IRM services not addressed by APIs or window resource management interfaces. In addition, these interfaces include elements such as protocol, look and feel, drivability, and style. This is the only interface that is used by the UI services to access the communication lines (indirectly through the Communications Services of the environment). Consequently, the HIDs associated with the EEI define the interface with the hardware that provides for correct execution of the EEI programs, but they do not include the interfaces to the communication lines; these are defined in the HIDs associated with the Communications Services.

Window Resource Management provides standard data formats to describe the attributes of graphical window system objects. Specifically, these interfaces provide the interaction between the UI services component and the data management services component of the IRM.

Interapplication Entities are internal UI service interfaces that support interoperability between applications and UI services software elements, and between different application programs based on the UI services.

Software Elements are programs that include the core functionality of the UI services, i.e., the X server. These programs play the major role in the interaction between all other services and the devices like the display, the keyboard and the mouse.

In addition to the UI services, Figure 9 shows an Application Program (on the right) and Other Services, which represents the rest of the services within the IRM. Figure 10 (integration matrix) shows relationships among the particular UI services, and specifically, which of the components interact directly. From this matrix we can see that the only way for an application (the client) to access the UI server (represented here as Software Elements) is either through the Application Program Interfaces or through the

Interapplication Entities. The Application Program Interfaces services do not have a direct connection in the integration matrix, since this communication takes place through the underlying X communication protocol rather than through a direct procedure call. The main role of the Interapplication Entities is to provide an interface and coordination for multiple applications interacting with the same X server. The server is shown to have direct interaction with the External Environment Interfaces. This is needed for both the interaction with the user through the hardware devices and with the communication lines. Figure 9 shows also that the Window Resource Management services interact with Other Services. The intent is to show that these services use the Data Management services to implement their functionality.

Services	Application Programs	Other Services	Application Program Interfaces	Inter-application Entities	Software Elements	Window Resource Mngmnt	External Environm. Interfaces
Application Programs			X	X			
Other Services						X	X
Application Program Interfaces	X						
Inter-application Entities	X				X		
Software Elements				X			X
Window Resource Mngmnt		X					
External Environm. Interfaces		X			X		

Figure 10: User Interface (UI) Services: Integration Matrix

In the rest of this appendix we first give a little more detail about the structure of the Application Program Interfaces. Then we briefly discuss all other groups of the UI services.

API Services

To show the structure of the API services we used the NIST OSE reference model [12] and represented it in Figure 11. In the IRM, this corresponds to the supporting document (hierarchy of subservices) described in Section 3.4. The layers of these services are described below.

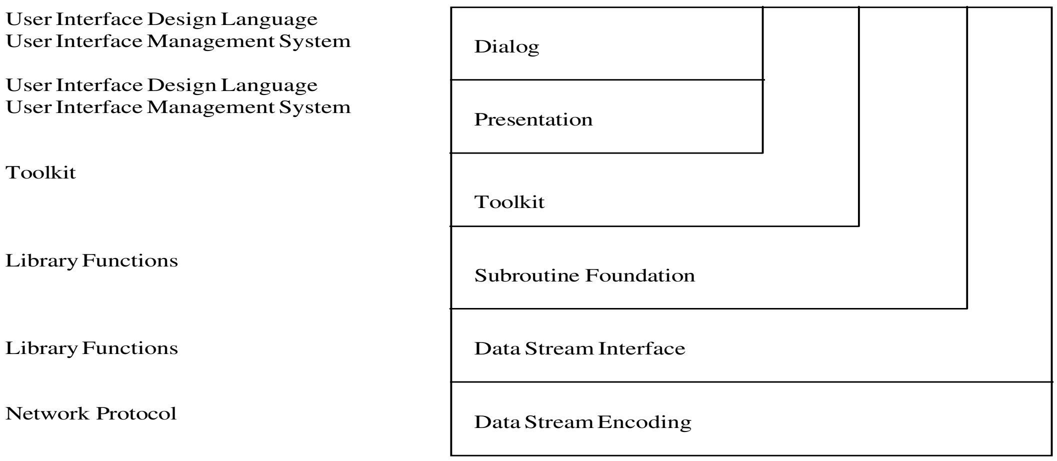

Figure 11: Application Program Interfaces: The Hierarchy of Subservices

Layer 0: Data Stream Encoding. This layer supports the format and sequencing for the stream of bytes exchanged between client and server processes. It is functionally equivalent to a networking protocol; it is called the X Protocol (see [12]).

The X protocol can be used over local interprocess communication channels between clients and servers running on the same machine as well as over a network connection. X is easily supported on numerous hardware platforms. However, to guarantee such a behavior both the software-to-software and software to hardware interfaces must be standardized. The standards for the software-to-software interfaces, within the IRM model, are specified in the SIDs, and the software-to-hardware interfaces are specified in the HIDs.

Layer 1: Data Stream Interface. This software interface, called the X Library, converts program-specified parameters to and from the data stream messages of layer 0. The specification of the X Library is given by [12].

The X Library API supports application portability by providing software developers with a single hardware-independent (provided that appropriate HDI specifications are fulfilled) graphics language supported on nearly all graphics workstations in the industry. As it can be seen in Figure 11, it is possible to write an application in which the user interface is implemented entirely in X Library calls.

Layer 2: Subroutine Foundation. These services, called the Xt Intrinsics, provide for creation, management, and destruction of user interface software objects. They also support event handling and management of the user interface state. The Xt Intrinsics is a collection of procedures written with the X Library for building and using widgets in application programs. Typical applications will use Xt along with a higher level, style-specific set of widgets. The specification of the Xt Intrinsics is also given in [12].

Layer 3: Toolkit Components. These services provide ready-made user interface components such as menus, buttons, and scroll bars. This layer includes the Virtual Toolkit, the Window System Toolkit, the Mapping Toolkit.

The Virtual Toolkit offers services like event processing (including mouse and keyboard handling), windows, graphics drawing, fonts, cursors, menu handling, pop-up dialogs, clipboard management. In lieu of the Virtual Toolkit API, the toolkit services may be satisfied by the native Window System Toolkits. These toolkits significantly extend the contributions of the Xt Intrinsics in the area of facilitating user interface software development and maintenance. They provide ready-made user interface components, commonly called widgets, such as scroll bars, buttons, and other controls easily integrated into an application.

Layer 4: Presentation. This layer determines the appearance of the user interface, such as which toolkit components are used, how they are organized, where they are positioned, and what colors are used. As shown in Figure 11, these services are not directly accessible to other services.

Layer 5: Dialog. This layer is responsible for the dynamic behavior of the user interface. It handles the communication between the presentation components (visible display objects) and the application code. It is responsible for interpreting user actions, like selection of a menu option, and making calls to the application code that will invoke the appropriate system response. It also determines the information currently displayed to the user and the form in which it appears. In addition, the Dialog layer handles all interaction with the user that does not affect an application's data or state, including zooming a graphics display and providing help to the user. Other services that can be provided at this layer include checking user inputs for validity and recovering from user errors. Finally, the Dialog layer insulates the application from screen management functions like window size and placement.

Unlike the Toolkit (and lower) layers of the reference model, it is not possible to implement the dialog layer without knowledge of the specific application with which it is to be used. Therefore, complete standardization of the services at this layer is not possible. However, there have been efforts to define, in an application-independent manner, the Application/Dialog layer interface and tools for specifying a particular Dialog layer implementation.

Layer 6: Application. This layer implements the specific functions required by the user by calling procedures at layers 1, 2, 3, and 5.

External Environment Interface (EEI) Services

External Environment Interfaces address both software interfaces to other IRM service components, including the Communication Services, and interfaces to the user. The UI services have an interface to the communication services through the basic window services and the X Server itself. The X Library and the X Server can be implemented to exchange X Protocol messages over any network protocol layers that provide bidirectional data transfer and delivery of bytes in sequence without duplication. They are the only UI service components that have direct interfaces with the communication services. ANSI H3X3.6 is working on defining a mapping of the X-Window System onto the OSI network services. For instance, TCP/IP and DECnet are two suites of networking protocols commonly supported by commercial implementations. As was mentioned earlier, the hardware specific elements of the communication protocols are defined in the IRM's HID associated with the Communication Services. The interaction among the layers of the communication protocol are specified in the appropriate SIDs.

Interfaces to the User are typically defined in a document called the style guide. A style guide contains guidance on the following topics: general user interface design principles, input and navigation models, selection and activation models, user interface component choices, layout, and interaction, window manager design, internationalization. Examples of user Interface style guides are the GPALS Human-Computer Interface Fixed and Mobile Standard [2] the DoD Human-Computer Interface (HCI) Style Guide [6] and the IEEE Recommended Practice for User Interface Drivability, IEEE P1201.2 [3]. Since the user interacts with the system through hardware devices, these kind of standards are part of the IRM's HIDs.

Window Resource Management

With respect to the IRM, these interfaces exist between the UI services component and the stored information component. The services include: X Resource Files and User Interface Language (UIL).

The X resource manager consists of a set of routines and database structures that maintain default values for resources. These default values are contained in a special type of file, called a resource file, which contains resource specifications in a human readable format. The syntax for resource specifications and the rules of precedence by which the resource manager processes them are given in [12].

UIL is a language that describes the initial state of a user interface. It can be used to specify the interface objects such as windows, menus, buttons, and controls, and the application procedures that must be called to support user interaction with those objects. The UIL compiler checks the interface specification for validity and generates a file that is read at run-time, thereby creating the widgets for the application.

Interapplication Entities

Inter-client communication services provide conventions to ensure that different applications based on the X-Window System can cooperate in the same environment. The X Protocol (described earlier in this appendix) supports this layer.

In the X-Window System's client/server architecture, multiple clients can share the services of the same server at any given time. While the X-Window System protocol defines the procedures for exchanges between a client and server, it does not guarantee that clients using the same server will run concurrently without interfering with each other. The Inter-Client Communications Conventions Manual (ICCCM) describes a minimal set of conventions at the protocol level to ensure that applications based on the X-Window System can cooperate in the same environment. Such conventions are described in [15, 16]. These conventions formalize guidance on design decisions that promote harmony between different applications coexisting in the same user interface environment.

Software Elements

The Software Elements services contain the basic windows services: X Server, Window Manager, Utility Programs. The server is the software that controls the display, keyboard, and pointing device. A valid X Server must accept X protocol requests and generate replies, errors, and events according to the X Protocol specification. The specification of the X Protocol is given in [12].

The X Server is the part of the X-Window System that insulates client applications from most differences in the underlying hardware associated with the user interface. The server handles all device dependencies where the actual manipulation of display hardware, keyboards, and pointing devices is performed. The interface between client applications and the server is the platform-independent X Protocol. The Window Manager is a special kind of client. It uses the window toolkit services and basic window services to interact with a display controlled by an X Server. Included in the software elements services are Utility Programs, i.e., programs that perform tasks to support other user-oriented functions. These include, for instance, hardcopy printing and runtime user event journaling.

Conclusions

This appendix shows an example of a group of services – the User Interface services – which are part of an Open System Environment. The services are modeled using the Integrated Reference Model (IRM) presented in this paper. This example shows the advantages of the IRM over other reference models according to the desirable features described in Section 2.3. Unlike any other model, the IRM clearly represents the relationships among particular services. Figure 9 shows the *part-of* relation – which services are part of the UI services. Figure 10 explicitly shows the *uses* (calling) relationships among the groups of services of the Open Environment and among the UI services themselves. For a selected group of services (API), Figure 11 shows both the *part-of* relation (which subservices are part of the APIs) and the *uses* relation. Every other group of services from Figure 9 could be represented in the same way as the APIs. Moreover, each of the layers of the API (and of any other services) could be refined using the same modeling primitives of the IRM. The IRM, through the combination of representation primitives, can accommodate any *uses* and *part-of* relationships. This is not possible within the other reference models, which either do not represent the *uses* relation (some of them represent only the *part-of* relation), or if they are committed to the strictly hierarchical representation, are not capable of representing non-hierarchical relations.

Additionally, by using the IRM, we were able to represent all the components of a computer system: software, hardware and the user. Specifications of interactions among these components have clearly defined places within the IRM so that the system specifier knows where to put or find such specifications. Again, this is a quite confusing issue in the other reference models.