--
--

IEEE P1003.0 Draft 14 – November 1991

their proper operation.

(8) Implementation Transparency

The POSIX OSE shall provide implementation technology transparency.

Rationale: The mechanism for implementation of services is not visible to the service user; i.e., only the service is visible to the service user.

(9) User's Functional Requirements

The POSIX OSE shall reflect the full scope of the user's functional requirements, within the context of the other requirements above.

Rationale: The POSIX OSE will provide the context within which application software portability can be addressed and it is the set of user's functional requirements that defines the scope of transportable service needs.

3.2 POSIX Open System Environment Reference Model

The POSIX OSE is based on a reference model with the full information system as its scope. As such, it spans the gap between requirement specification and the design of any specific information system. The reference model provides a set of conventions and concepts, mutually agreed upon between the information system user and provider communities. This common understanding is key to achieving application software portability, system interoperability, and may encourage software reuse. It will certainly allow for more compact and correct procurement specifications.

The definition of this reference model is an engineering and management task and not a scientific one. There are many possible models and, while it might be interesting to contemplate an optimal one, a reference model that satisfies the requirements is all that is necessary.

An information system reference model must satisfy conflicting requirements similar to those encountered in traditional architectural disciplines. The reference model must be structured enough to encourage the generation and use of standards and standard components. Yet it must also be flexible enough to accommodate tailored and special purpose components necessary to meet realworld needs.

The POSIX OSE Reference Model is a set of concepts, interfaces, entities, and diagrams that provides a basis for specification of standards. The POSIX OSE Reference Model will provide guidance and direction for future standardization and integration efforts. In order for the POSIX OSE to evolve and mature, it will be necessary for the reference model to provide insights into those services and capabilities for which standards do not currently exist and for which appropriate standardization activities cannot be identified.

The POSIX OSE Reference Model is described from the user perspective; i.e., the reference model records the application platform user's perception (mental model) of the overall large distributed system used to support the user enterprise. This point of view will assure that the:

— Information technology users will have the proper services to meet their requirements, and

— Information technology vendor implementations will not be constrained unnecessarily.

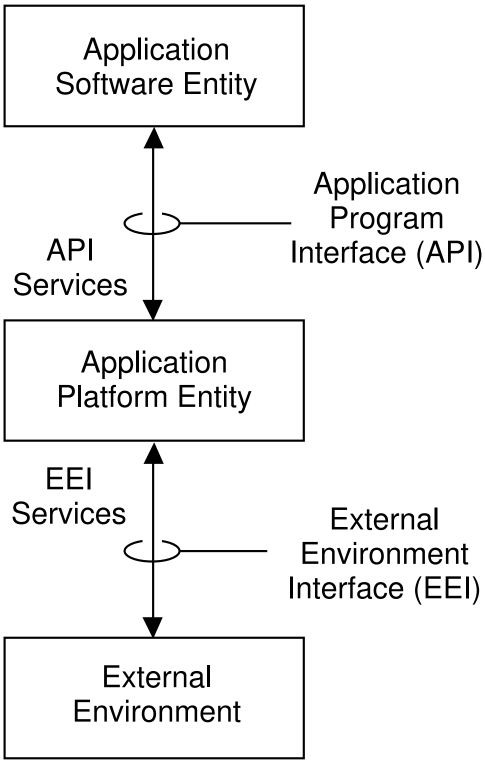

Figure 3-1 – POSIX OSE Reference Model

Figure 3-1 depicts the basic elements of the POSIX Open System Environment Reference Model. These include three entities (Application Software, Application Platform, and External Environment) and two interfaces between them, identified as the Application Program Interface (API) and the External Environment Interface (EEI). The application platform provides API and EEI services across the associated interfaces.

This model has been generalized to such a degree that it can accommodate a wide variety of general and special purpose systems. More detailed requirements exist for each service category described in Section 4. The service specification has been defined to be robust and flexible enough to allow subsets or extensions for each category as needed. As a result, the POSIX OSE reference model is able to accommodate a variety of architectures and standardization approaches. It should be possible to show where any relevant standard fits within the reference model.

Standards (in the sense of formally adopted consensus specifications) address only interfaces between entities, as well as services and supporting formats offered across those interfaces. The interface specification defines a convention adopted to represent the function offered across the interface in both directions. Note that no set of standards can, by itself, assure portability of specific applications. Applications must be properly engineered with an explicit portability objective in order to achieve it.

The Reference Model is not a layered model. The application platform provides services to a variety of users across both platform interfaces. A human being invokes the platform services at the External Environment Interface. If an application developer is the application platform user, the services offered at the application program interface (API) are invoked at the source code level.

All of these features may be available locally or remotely if the system is connected to a larger distributed system. All other resources and objects can be conceptualized as being contained within the application platform.

Note that the actual implementation of any given system element may differ greatly from the reference model presented. The intention is to define a conceptual reference model that the widespread design, implementation, and integration communities may assume in executing their activities. Partitioning of function for purposes of discussion or specification does not imply or endorse similar partitioning for design or implementation.

3.2.1 Reference Model Entities and Elements

Figure 3-2 expands Figure 3-1 to identify elements of the Reference Model entities. For the purposes of this discussion, the term ''entities'' will be used when discussing the classification of items (i.e., ''things'') related to application portability. The term ''component'' will only be used when an entity is further decomposed into constituent parts. The application software entity is the only entity that is decomposed into components.

Application Software is defined (see 2.2.2.4) as software specific to an application. It is composed of:

— Programs (source code, command/script files, etc.)

— Data (user data, application parameters, screen definitions, etc.), and

— Documentation (online documentation only; hardcopy not included).

An application program is represented by source code, produced according to a specific programming language and a set of language bindings (i.e., API specifications) for the required services. These specifications may be public standards or other open specifications.

An application program may be divided into two parts:

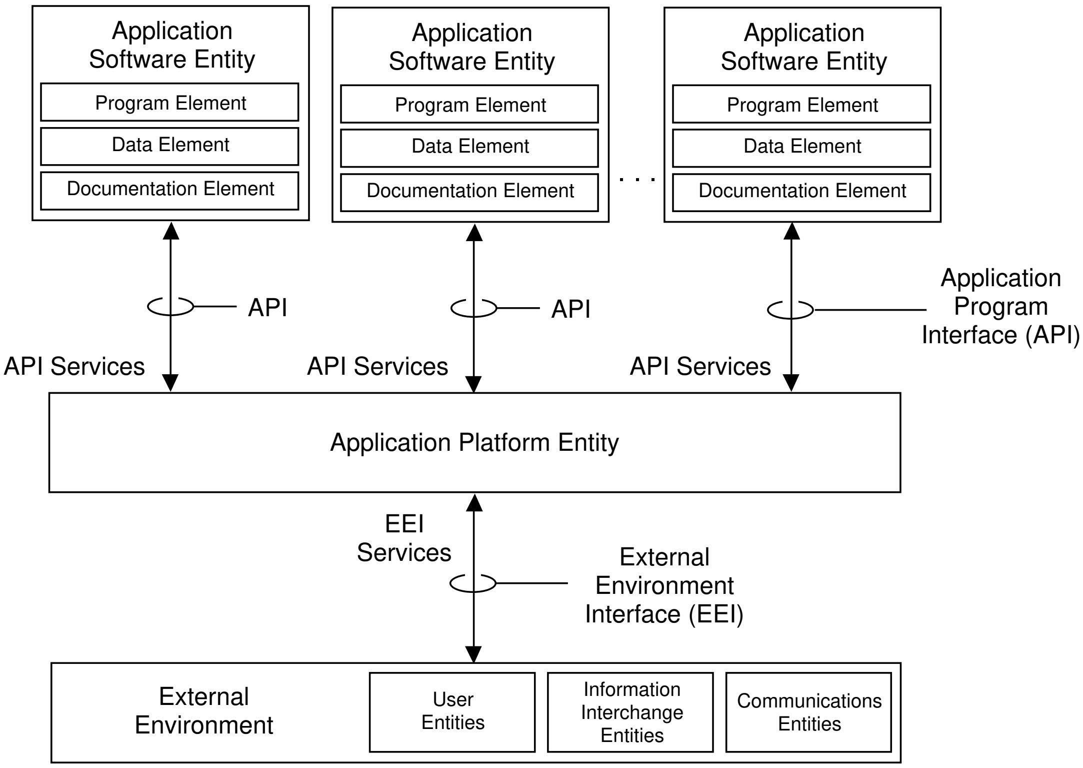

Figure 3-2 – POSIX OSE Reference Model — Entities

— An *invariant* portion of source code, requiring no change when ported, and

— A *variant* portion of source code, which requires changes when ported.

The objective of any effective application software portability method should be to minimize the ‘‘variant’’ portion of the application software via creation and use of API standards. This would ideally allow application software components to be moved to a different (but portability-standard compliant) system and run without source code modification. However, since standards exist for which strictly conforming application software requires modification (e.g., memory requirements, processor-specific COBOL statements), this can only be approximated.

Separate but related standards may be required to support the portability of each of the elements listed above. Examples of application software are the familiar word-processing, spreadsheet, or accounting packages, as developed by the consumer or a commercial application software developer. Each of these packages appears as an application software entity when executed on an application platform.

One or more applications may run on a given application platform simultaneously, as represented by the boxes at the top of Figure 3-2. Each application can be thought of as an independent application entity, communicating and synchronizing with other applications, if necessary, via a variety of communications mechanisms.

The Application Platform is defined (see 2.2.2.2) as the set of resources that support the services on which an application or application software will run. It provides services at its interfaces that, as much as possible, make the implementation-specific characteristics of the platform transparent to the application software.

In order to assure system integrity and consistency, application software entities competing for application platform resources must access all resources via service requests across the API. Examples of application platform elements could include an operating system kernel, a realtime monitor program, and all hardware and peripheral drivers.

The application platform concept does not imply or constrain any specific implementation beyond the basic requirement to supply services at the interfaces. For example, the platform might be a single processor shared by a group of applications, or it might be a large distributed system with each application dedicated to a single processor. (See 3.2.4.)

The application platform for systems built to the POSIX OSE will differ greatly depending upon the requirements of the system and its intended use. It is expected that application platforms defined to be consistent with the POSIX OSE will not necessarily provide all the features discussed here, but will use tailored subsets for a particular set of application software.

The External Environment contains the external entities with which the application platform exchanges information. These entities are classified into the general categories of human users, information interchange entities, and communications entities.

Human users are not further classified, but are treated as an abstract, or average, person. Information interchange entities include removable disk packs, floppy disks, and security badges. Communications entities include phone lines, local area networks, and packet switching equipment

3.2.2 Reference Model Interfaces

Figure 3-3 expands Figure 3-1 to identify the services available at the reference model interfaces.

Between these three classes of entities there are two types of interface where standards and other open system specifications are required to enable application software portability and interoperability. These two interface types are labeled as the Application Program Interface (API) and the External Environment Interface (EEI).

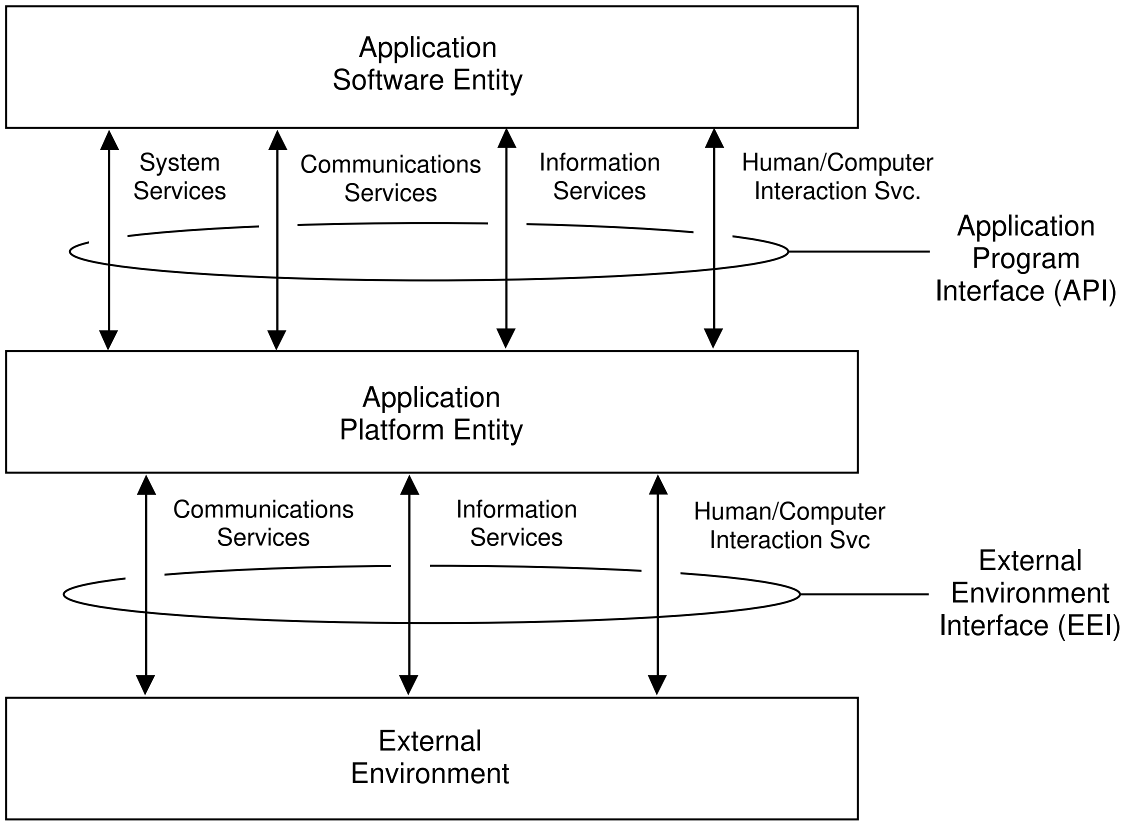

Figure 3-3 – POSIX OSE Reference Model — Interfaces

3.2.2.1 External Environment Interface (EEI)

The External Environment Interface is defined (see 2.2.2.8) as the interface between the application platform and the external environment across which information is exchanged. It is defined primarily in support of system and application software interoperability. User and data portability are directly provided by the EEI, but application software portability also is indirectly supported by reference to common concepts linking specifications at both interfaces. The services available at the EEI comprise:

- Human/Computer Interaction Services
- Information Services
- Communications Services

The Human/Computer Interaction EEI is the boundary across which physical interaction between the human being and the application platform takes place. Examples of this type of interface include CRT displays, keyboards, mice, and audio input/output devices. Standardization at this interface will allow users to access the services of compliant systems without costly retraining.

The Information Services EEI defines a boundary across which external, persistent storage service is provided, where only the format and syntax is required to be specified for data portability and interoperability.

The Communications Services EEI provides access to services for interaction between internal application software entities and application platform external entities, such as application software entities on other application platforms, external data transport facilities, and devices. The services provided are those where protocol state, syntax, and format all must be standardized for application interoperability.

3.2.2.2 Application Program Interface (API)

The Application Program Interface (API) is defined (see 2.2.2.3) as the interface between the application software and the application platform across which all services are provided. It is defined primarily in support of application portability, but system and application software interoperability also are supported via the communications services API.

The POSIX OSE API is a combination of a number of standards-based interfaces. It can be thought of as a bookshelf containing several standards-based APIs, with each API a separate book on the bookshelf.

The POSIX OSE API specifies a complete interface between the application software and the underlying application platform, and may be divided into the following parts:

— System Services API E

— Communications Services API E

— Information Services API E

— Human/Computer Interaction Services API E

The last three APIs listed are required to provide the application software with access to services associated with each of the external environment entities. E

The first API is required to provide access to services associated with the application platform internal resources, identified as the System Services API. This interface may be divided into two types of specifications; i.e., Language Service and System Services API specifications. E

Definitions of services at the API take the form of programming-language specifications, language-independent service specifications, and language bindings for the service specifications. These specifications may be described as follows:

(1) Those traditionally associated with the language specifications, such as program control (if ... then ... else), math functions, string manipulation, etc., defined as *the programming language API*, and

(2) Services provided by the underlying application platform defined independent of language, such as interprocess communications,

interobject messages, access to the user interface, and data storage. Specifications of for these services are defined independently of any programming language, and are identified as *language-independent service specifications* .

(3) The language-independent service specifications are translated into language-specific specifications used by programmers in writing applications. These specifications provide access to the services using methods consistent with a specific programming language. Such language-specific specifications are called *language-binding API s*.

Creation of a *language-independent service specification* facilitates the management and development of consistent language binding standards. The language-binding specifications are used directly by programmers and application platform suppliers in implementing application software and platforms.

The ''programming language''/''language binding'' dichotomy may be a result of the way Information Technology standards are currently developed. Programming language specifications are developed with the goal of being ''system independent'' (e.g., C, COBOL , FORTRAN , etc.). Language Binding specifications (e.g., POSIX .1 {2}, MOSI , etc.) are being translated into ''language-independent'' specifications, with one or more bindings for specific languages.

3.2.3 EEI -API Service Relationships

The relationships between similarly named services provided at the API and the EEI are not simple one-to-one relationships. For example, a data storage service interface may provide an application with transparent access to a remote file via network services. In this case, the completion of the data storage service provided at the API is dependent upon, and can be thought of as having been ''translated'' into, communication services provided at the EEI.

Fortunately, it is not essential for the purpose of satisfying the requirements of the POSIX OSE to specify these relationships in detail. In fact, a detailed definition could unnecessarily constrain the implementation. A given implementation of the application platform will define the relationship between the API and EEI in different ways.

3.2.4 POSIX OSE -Based Distributed Systems

In a distributed environment, multiple application platforms may interact by way of a network external to the platforms, but connected to them via the communications EEI , as in Figure 3-4. For an application software entity to gain access to the EEI services, communications services are requested at the API. The implementation of the application platform translates these API requests into appropriate action at the EEI.

Communication occurs between application platforms via external entities that implement the data transport function. These can use a wide variety of implementation methods and protocols, providing access to distributed data and

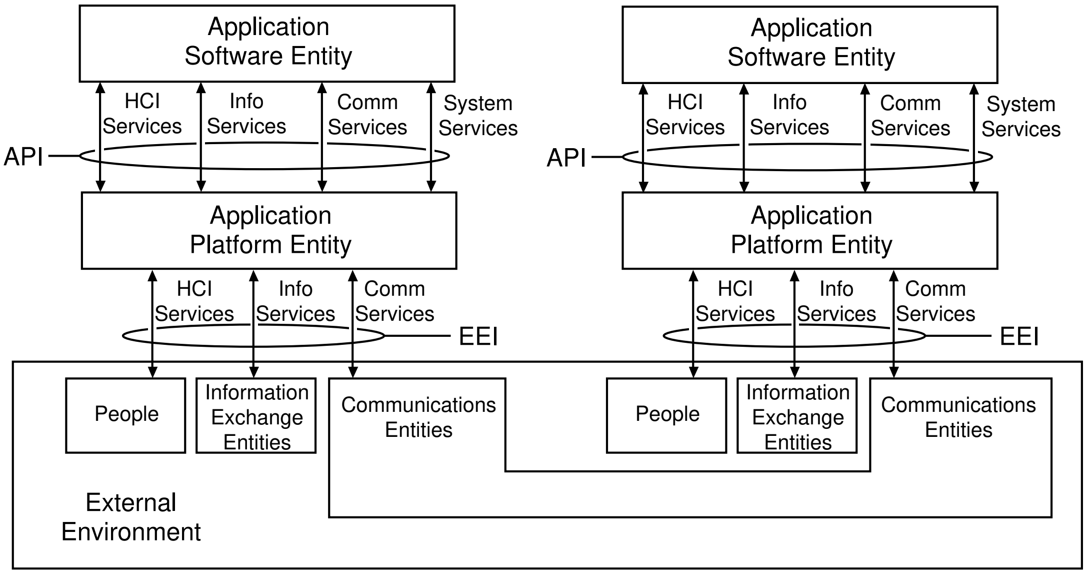

Figure 3-4 – POSIX OSE Reference Model — Distributed Systems

services via the network.

Distributed Systems are manifest in this model primarily through the use of the distributed system network services API. As can be seen in Figure 3-5, distributed systems are a refinement of the POSIX Network Environment Model shown in Figure 4-3. As such, a perceived Application Platform may in fact be comprised of several (or many) individual application platforms. However, in the distributed environment, they operate and are viewed as a single entity by the using applications. Within this extended application platform are the embedded network services necessary for the elements of a distributed environment to function.

Within the distributed environment, network access between the platforms that make up the ‘‘perceived’’ application platform are handled using the Distributed Systems Network Services APIs. Network services for access between ‘‘perceived’’ application platforms will use the Network Services EEI between the platforms.

3.3 POSIX Open System Environment Services

This guide defines a uniform set of standard services provided to users of application platforms in support of POSIX objectives of application portability and system interoperability. These services are available to users across specified interfaces keyed to the POSIX reference model defined in 3.2.

The POSIX OSE services are divided into categories described by the clauses in Section 4. Each category begins by defining a more detailed and specialized version of the OSE reference model (see 3.2) to provide context for service