NISTIR 5736

Comparison of POSIX Open System Environment (OSE) and Open Distributed Processing (ODP) Reference Models

Geraldina Fernandes
Joseph Hungate

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Distributed Systems Engineering
Gaithersburg, MD 20899

NIST

Computer Systems Laboratory
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce
Gaithersburg, MD 20899

Comparison of POSIX Open System Environment (OSE) and Open Distributed Processing (ODP) Reference Models

Geraldina Fernandes
Joseph Hungate
Distributed Systems Engineering

November 13, 1995

National Institute of Standards and Technology
Computer Systems Laboratory

Table of Contents

1 Introduction

Advances in technology, giving increasing importance to information service networks, and personal workstations are two factors which permit the construction of distributed applications running on a large set of interconnected systems. With the increasing need to interconnect information processing systems, distributed systems are becoming very important. Organizations need to exchange information, both inside the organization and between cooperating organizations.

Many organizations are becoming dependent on distributed computing systems. When a distributed processing system stops providing the service required by its users, an entire factory's production may stop, a bank cannot conduct its normal transactions or an airline cannot make reservations. Management of these heterogeneous software and hardware environments, in order to supply and maintain the services required, is one of the most difficult problems facing computer users today.

Management services provide mechanisms to monitor and control the operation of individual applications, databases, systems, platforms, networks, and user interactions with these components. They also enable users and systems to become more efficient in performing required work. Standards for management services and an integrated approach to assure consistency are just now being addressed by standards development organizations and user consortia.

This report presents two existing reference models for describing distributed systems. These models are the Open Distributed Processing reference model and the POSIX OSE reference model. They both address distributed open systems, but from different perspectives, with different objectives and methodology. However, there are some similarities between these two models and the problems they try to solve may be considered complementary.

After a brief description of the two models, a comparison is made referring in particular to the extent distributed systems management is accommodated in both models.

2 Open Distributed Processing (ODP) Overview

The objective of ODP standardization is to develop standards that allow the distribution of information processing services in an environment of heterogeneous information technology resources and multiple organizational domains[1].

ODP standardization aims to build distributed systems which include the following properties:

- Openness: This property is achieved through both the contribution of portability (the ability to execute different system components on different processing nodes without modification) and interworking (the ability to support meaningful interactions between components, possibly residing in different systems).
- Integration: This property allows incorporation of various systems and resources (e.g. systems with different architectures, or different resources with different performance) into a single system. This helps to deal with heterogeneity.
- Flexibility: This property consists of supporting a system's evolution. A system should be capable of facing run-time changes and be, for instance, capable of being dynamically reconfigured to accommodate those changes.
- Transparency: This is a central requirement to facilitate distributed applications construction, hiding the distribution details from the developers.

2.1 The Reference Model of ODP

The Reference Model of ODP (RM-ODP) provides a framework for the standardization of Open Distributed Processing. It creates an architecture within which support for distribution, interworking and portability can be integrated.

The purpose of the RM-ODP framework is to organize services within autonomous systems in order to facilitate interworking of software components distributed into progressively larger and larger systems.

The RM-ODP is organized into two main parts:

- A descriptive model, found in volume 2 of the standard, of distributed systems. It takes an object-based modeling approach that provides a common ground for all the ODP viewpoint specifications.
- A prescriptive model, found in volume 3 of the standard, that describes the organization

and structure of an open distributed system. It is organized around the ODP five viewpoints.

2.2 Distribution Transparency

Distribution transparencies are used to hide aspects of ODP systems that arise through their distribution. The ODP infrastructure supports a set of distribution transparencies:

- Access transparency: Masks differences in data representation and invocation mechanisms.
- Failure transparency: Masks, from an object, the failure and possible recovery of other objects (or itself), to enable fault tolerance.
- Location transparency: Masks the use of information about location in space.
- Migration transparency: Masks, from an object, the ability of a system to change the location of that object. Migration is often used to achieve load balancing and reduce latency.
- Relocation transparency: Masks relocation of an interface from other interfaces.
- Replication transparency: Masks the use of a group of objects with compatible behaviors to support an interface. Replication is often used to enhance performance and availability.
- Persistence transparency: Masks, from an object, the deactivation and reactivation of other objects (or itself).
- Transaction transparency: Masks coordination of activities amongst a configuration of objects to achieve consistency.

Distribution transparency is selective. Application designers do not need to be aware of those aspects of distribution. When addressing the distribution of their applications, they only have to express their requirements for transparencies.

2.3 Viewpoints

The complete specification of a distributed system usually involves a very large amount of information, making it difficult to capture all aspects of the design in a single description. The RM-ODP describes five viewpoints from which an ODP system can be described. Each viewpoint reflects a different set of requirements and concerns which arise in the specification of an ODP system. These viewpoints are not independent. Each one is a partial view of the complete system specification. Some items can therefore occur in more than one viewpoint.

The five viewpoints defined in the RM-ODP are:

- Enterprise viewpoint: Is concerned with the business activities of the specified system. It focuses on the purpose, the scope and the policies for the system.
- Information viewpoint: Is concerned with the information that needs to be stored and processed in the system. It focuses on the semantics of information and of information processing.
- Computational viewpoint: Is concerned with the description of the system as a set of objects which interact at interfaces, without considering the details of how they interact.
- Engineering viewpoint: Is concerned with the mechanisms and functions required to support distributed interaction between objects in the system.
- Technology viewpoint: Is concerned with the detail of the components from which the distributed system is constructed. It focuses on the choice of technology in that system.

The RM-ODP defines a viewpoint language for each viewpoint. Each viewpoint language defines concepts and rules for specifying ODP systems from the corresponding viewpoint. A set of specifications of an ODP system written in different viewpoint languages should be mutually consistent, (i.e., should not make mutually contradictory statements)[2].

2.4 ODP Modeling concepts

Objects and actions are the most basic ODP modeling concepts. All things of interest are modeled as objects and anything of interest that can happen is an action.

Objects are entities that contain information and offer services. An object is characterized by its identity and by encapsulation, abstraction and behavior.

The fact that information contained in an object is accessible only through interactions at the interfaces the object supports is termed *encapsulation*. The object's internal operation is not visible at the object boundary, unless its attributes, operations, or notifications are defined to expose that information.

Abstraction implies that the internal details of an object are hidden from other objects. This property is crucial to deal with heterogeneity, permitting different services to be implemented in different ways, using different mechanisms and technologies, thus enabling portability and interoperability. Object abstraction is also essential to support system's evolution. It builds a strong separation between objects, enabling them to be replaced or modified without changing their environment, as long as they continue to support the services the environment expects.

The *behavior* of an object is the set of all potential actions in which that object may take part. The state characterizes the situation of an object at a given instant.

An *interface* is the only means to access an object. An ODP object may have many interfaces, for functional separation and distribution purposes. Each interface is an abstraction of the object's behavior that consists of a subset of the interactions of that object together with a set of constraints on when they may occur[3].

Templates are used to describe objects. Objects with the same behavior may be described by the same template.

A *type* is a predicate, (i.e., a property of a collection of objects. Objects do not have to be similar to satisfy the same type, they only need to have the properties prescribed by the type. A *class* is the collection of objects associated with a type. Type A is a *subtype* of a type B, and B is a supertype of A, if every object which satisfies A also satisfies B. Class A is a subclass of another class B, and B is a superclass of A, when the type associated with A is a subtype of the type associated with B[3].

2.5 ODP Architecture

The prescriptive model[2] makes statements which must hold for a system to be considered as an ODP system. It presents the ODP Architecture, which consists of a set of rules to define the structure of an ODP system and the interrelationships between its parts.

The Reference Model defines a framework comprising:

- Viewpoints.
- A viewpoint language for each viewpoint.
- Specifications of the functions required to support ODP systems.
- Transparency prescriptions showing how to use the ODP functions to achieve distribution transparency.

The architecture for ODP systems and the composition of functions is determined by the combination of the computational language, the engineering language and the transparency prescriptions.

2.5.1 Computational language

The computational language provides the concepts that can be used to structure distributed applications. In the computational viewpoint, an ODP system is specified as a set of interacting objects providing application specific functions. The details of the infrastructure which supports these interactions are not visible in the computational viewpoint. They are described in the engineering viewpoint.

Each interaction of a computational object occurs at one of its interfaces. The Reference Model specifies signature type rules for computational interfaces. Signature subtyping rules define minimum requirements for one interface to substitute another.

For two objects to co-operate, they must be bound to each other. The ODP computational language supports both implicit and explicit binding. In implicit binding the objects are bound directly (see figure 1).

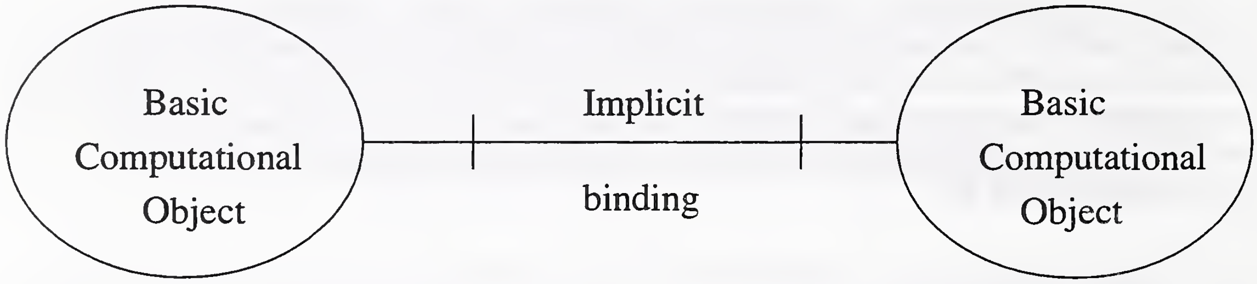

Figure 1 Implicit Binding

One object obtains an identifier to an interface of another by some means, either from a trader or as a result of some other interaction. At some unspecified time between receipt of the interface identifier and the first invocation of an operation in the interface, a binding process occurs.

Explicit binding is performed as an action. A binding object is created, which encapsulates the binding mechanisms, allowing the binding to be manipulated (see figure 2).

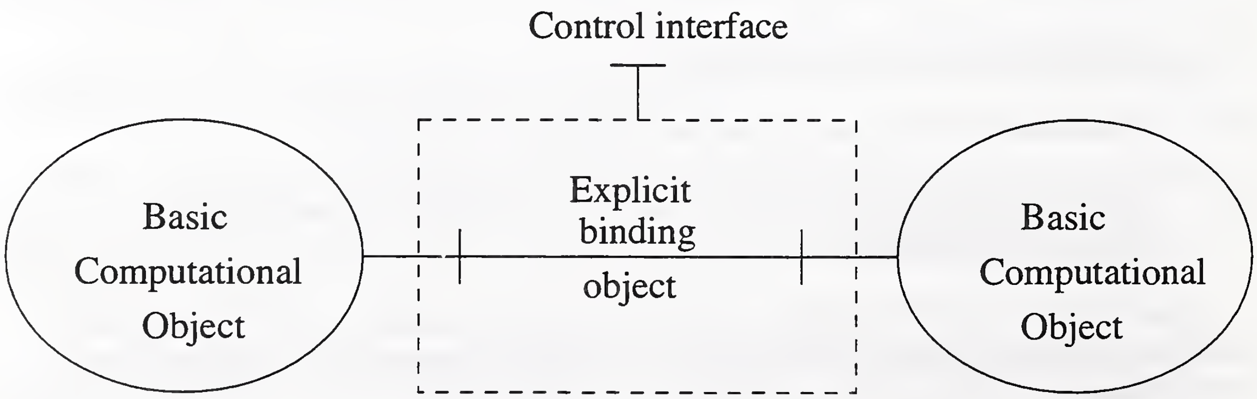

Figure 2 Explicit Binding

The control interfaces of a binding object provide functions such as: monitoring the binding, authorizing changes to the binding, changing the quality of service of the binding, deleting the binding.

2.5.2 Engineering language

The engineering language provides concepts for describing the infrastructure required to support distributed interaction between objects in an ODP system. It contains rules for:

- Structuring communication channels between objects (stubs, binders, protocol objects, interceptors).
- Structuring systems for the purpose of resource management (node, nucleus, cluster, capsule).

The Basic Engineering Objects (BEO) correspond to Basic Computational Objects (BCO) (see figure 3)[1]. BEOs are grouped into *clusters* that represent, for instance, a piece of C code. Clusters are organized into a *capsule* that can be compared with a UNIX process. The capsule is bound to a *nucleus* which represents, for instance, a particular operating system. A nucleus belongs to a *node*, representing for instance a workstation.

The refinement of computational templates into engineering templates can be compared to compiling programs to produce object code. The refinement of engineering templates into cluster templates corresponds to linking modules.

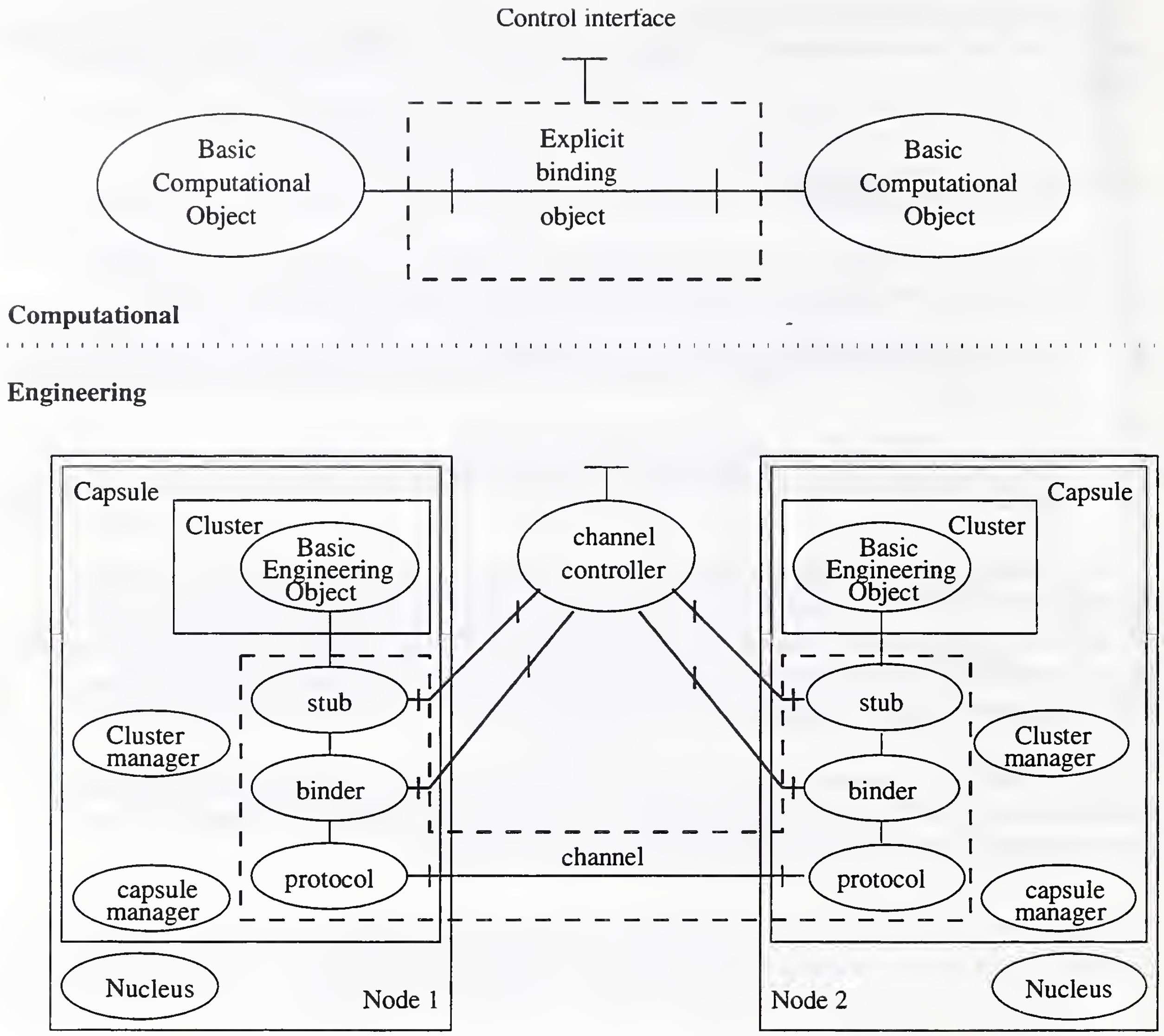

Figure 3 Computational and Engineering Correspondence

A *channel* supports distribution transparent interaction between BEOs. It is a configuration of stubs, binders, protocol objects and interceptors, interconnecting a set of BEOs. A computational binding corresponds to a channel configuration in the engineering specification.

Stubs, binders, protocol objects and interceptors are all engineering objects.

BEOs which interact via channels (BEOs in the same cluster do not require channels to interact) are locally bound to stubs. A stub provides the conversion of data carried by interactions. It can apply controls and keep records (e.g., for security and accounting) and it can also have a control interface for quality of service management.

Binders interact with one another to maintain the integrity of the binding. They provide relocation transparency by monitoring communication failures and repairing broken distributed bindings. A binder may have a control interface, which enables changes in the configuration of the channel and deletion of the channel.

Protocol objects provide communication functions, assuring that computational objects can interact remotely with each other. When protocol objects in a channel are of different type, they require an *interceptor* which provides protocol conversion. All protocol objects in a communication domain can communicate directly. An interceptor stands at the boundary between domains and provides checks and transformations on interactions that cross domain boundaries.

2.6 Management Functions

Resource management requires that it be possible to invoke management operations on individual services, the objects that contain them, the cluster that contains the object, the capsule that contains the cluster and the node that supports the capsule.

A cluster is a set of BEOs associated with a cluster manager. Each BEO in a cluster can be bound to other BEOs in the same cluster, or in other clusters. Additionally, a BEO can have an interface supporting the object management function. This object interface is bound to the cluster manager. A BEO is also bound to the nucleus at an interface providing the node management function.

Each cluster manager in a capsule is bound to the capsule manager for that capsule. This structure is presented on figure 4[2].

The cluster management function can checkpoint, recover, migrate, deactivate and delete the cluster. It may require the cluster manager to interact with the objects in the cluster via their object management interface. Cluster instantiation is performed by a capsule manager.

A capsule consists of: cluster(s), cluster managers (one for each cluster) and a capsule manager. The capsule management function instantiates clusters from a cluster template and can checkpoint, deactivate and delete a capsule. Capsule instantiation is performed by the nucleus.

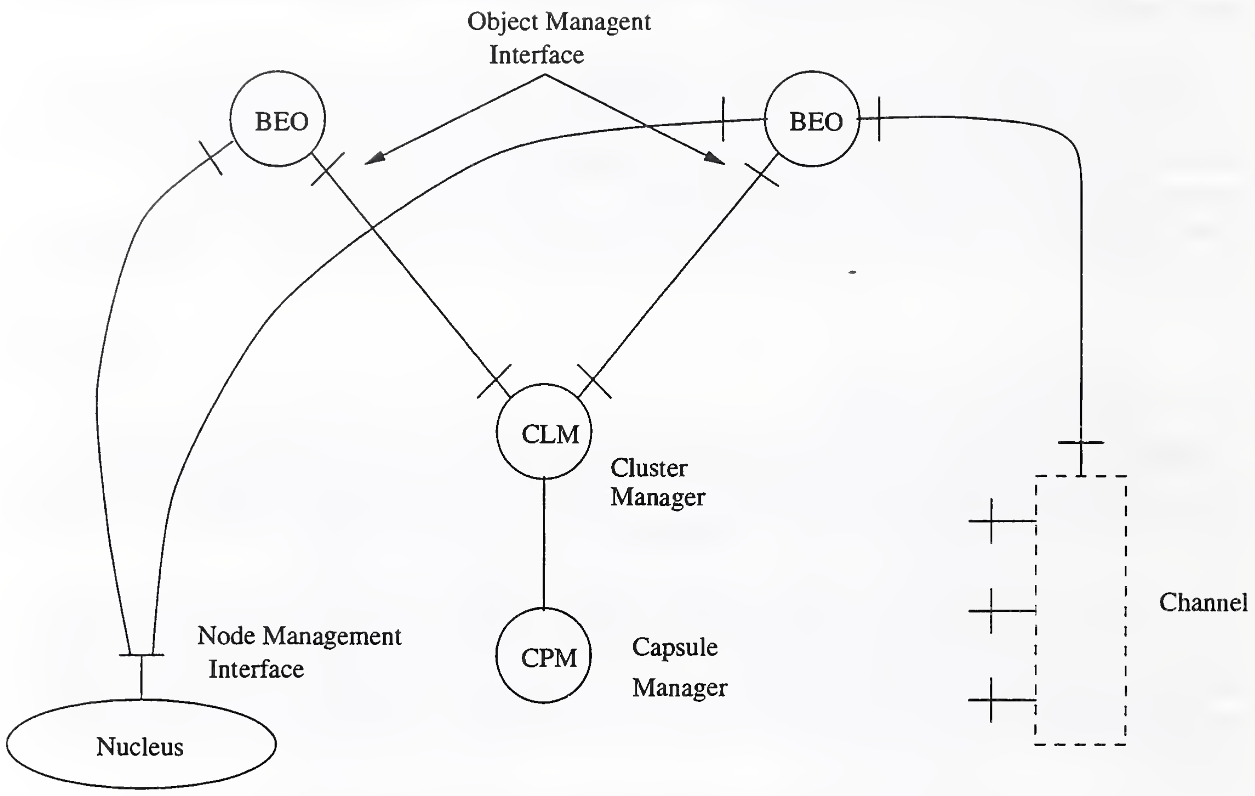

Figure 4 Structure supporting a basic engineering object

A node consists of one nucleus and a set of capsules. The nucleus provides a set of node management interfaces, one to each capsule within the node. The node management function provides thread management for objects which can provide spawning, forking and joining of activities defined in the computational language.

2.7 Management of ODP Systems

An ODP system consists of a number of ODP applications which make use of underlying support services. All these services, as well as the ODP applications, need to be managed. Management in ODP is concerned with overall systems management, including application management and communication management.

A managed object is an entity to which a management policy applies and which behavior can be monitored and changed by a manager. It is an abstraction that represents the properties of data processing and data communications resources, for the purposes of management. A managed object could be a hardware or software component, or a collection of information (e.g. a data structure).

Managers monitor the activities of managed objects, make management decisions based on that information and perform control actions on the managed objects, modifying the system's behavior.

In an ODP system, only an object can modify its own behavior. In order to realize ODP management applications, it is necessary to specify management interfaces supporting the management operations that can be performed on a managed object. Thus, a managed object, besides its normal functional interfaces, can provide more than one management interface reflecting different management views.

There are three types of interactions between managers and managed objects:

- Control actions: are directed by the manager to the managed object.
- Requests for information: are issued by the manager and result in a reply from the managed object sending the required information.
- Notifications: are sent to the manager by the managed object to report faults or events.

Within the ODP environment there are multiple coexisting management views and boundaries of responsibility. To reflect these different views, *domains* provide a means of specifying boundaries of management responsibility and authority. All the managed objects in a domain are controlled under a common management policy. An important aspect of a management policy is to specify what management operations managers may perform on the objects they manage. Management policy and domain membership can be modified through interactions with a single object, called the domain coordinator managed object. This way managers do not need to interact individually with the multiple managed objects within the environment.

Domains do not encapsulate the member objects, external objects may interact directly with an object in a domain.

Domain relationships can be used to model management structures. It is necessary to permit different domains, with members in common, to coexist in order to reflect the different required views of management. Two domains are disjoint if they have no member objects in common. Two domains overlap if there are objects which are members of both domains. The necessary transformations in order to allow activities to cross from one domain to another take place at the federation boundaries.

3 POSIX Open System Environment (OSE) Overview

An Open System Environment (OSE) is a computing environment which encompasses the functionality needed to provide interoperability, portability and scalability of computerized applications distributed across networks of heterogeneous, multivendor hardware/software/communications platforms[4].

Interoperability, portability and scalability are the three major concepts of open systems:

- Interoperability: the capacity of systems to exchange information and to mutually use the information that has been exchanged.

- Portability: the ability of application software source code and data to be transported without significant modification to more than one type of platform or operating system.

- Scalability: the ability to move, without significant modification, application software code and data into systems and environments with a variety of performance characteristics and capabilities, ranging from standalone microcomputers to large heterogeneous distributed systems.

3.1 OSE Reference Model

The Institute of Electrical and Electronics Engineers (IEEE) POSIX Working Group P1003.0 describes an OSE Reference Model (OSE/RM), which provides a framework for describing open system concepts and defining a terminology that can be agreed upon by all interested parties. The OSE/RM spans the gap between system requirements specification and design of a specific system that uses information technology (IT)[5], allowing future information system users (who state requirements) and providers (who provide the system to fulfill those requirements) understand each other.

The OSE/RM provides a basis for specification of standards to enable application portability and interoperability. It describes an OSE identifying system entities and key interfaces between them that are related to application software portability and interoperability (see figure 5).

The five classes of entities identified by the OSE/RM are:

- Application Software Entities: software specific to an application. It is composed of one or more of: programs, data, documentation and training.

- Application Platform Entity: comprises the collection of hardware and software components that provide the system services used by the application software. It provides services at its interfaces.

- User: users are not further classified, they are treated as typical persons.
- Information Interchange Entities: physical data storage media, including, for example, CD-ROM, magnetic tapes, floppy disks and security badges.
- Communication Entities: include external data transport facilities and devices, such as phone lines, local area networks and packet switching equipment.

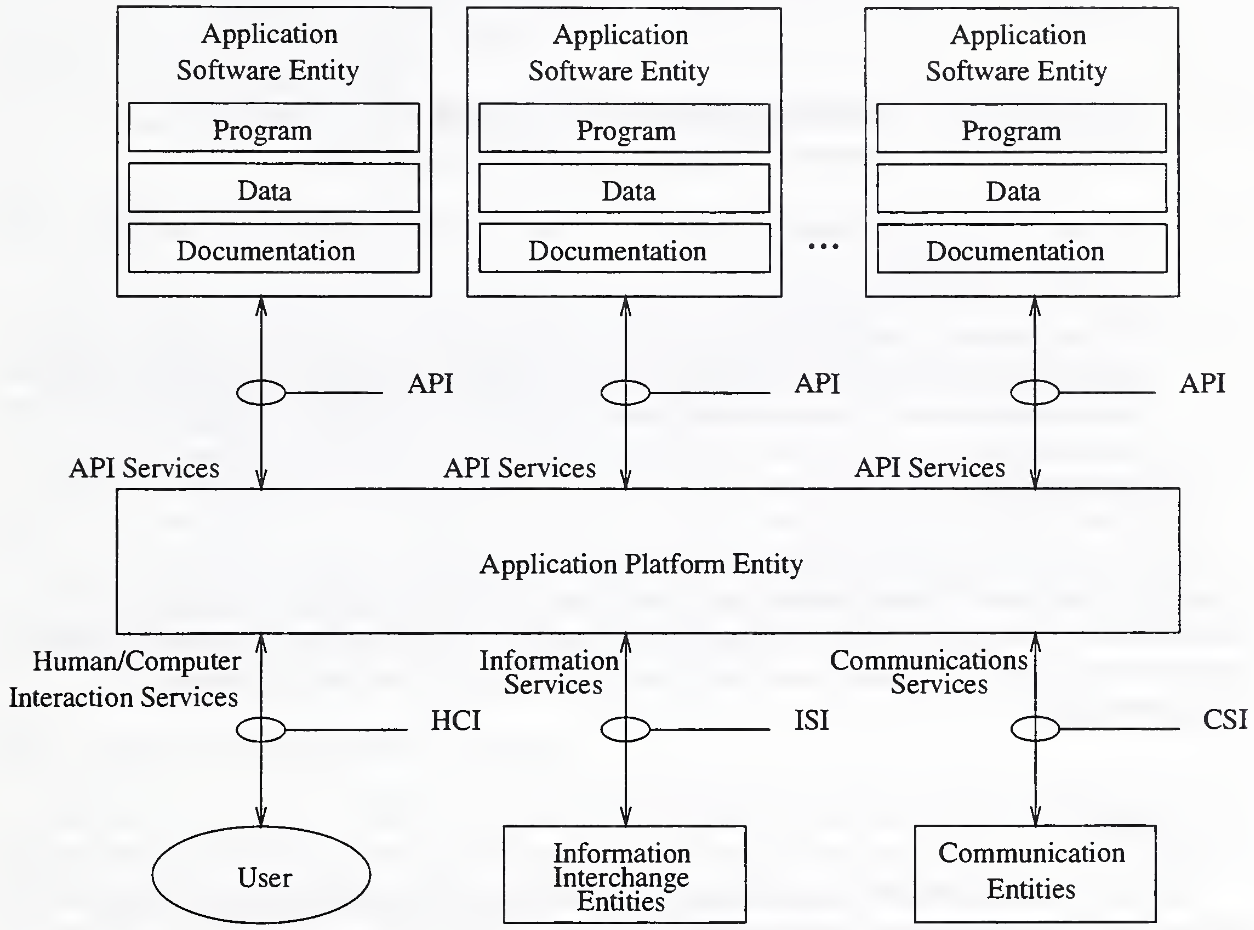

Figure 5 POSIX OSE Reference Model

There are four classes of interfaces in the OSE/RM:

- Application Program Interface (API): the interface between the application software and the application platform. Its primary function is to support portability of application software. There are four types of services accessible via the API:

  - Internal system services
  - Human/Computer interaction services
  - Information interchange services
  - Communication services

  Internal system services provide access to services associated with resources that are internal to the application platform. The other three types of services listed provide the application software with access to services associated with external environment entities.

- Human/Computer Interface (HCI): the boundary across which physical interaction between the human user and the application platform takes place.

- Information Services Interface (ISI): the boundary across which external, persistent storage is provided.

- Communication Services Interface (CSI): 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. This interface supports system and application software interoperability.

The API, HCI, ISI and CSI define the services provided by the application platform to its users. The definition of these interfaces must be driven by the user requirements and be as independent of the implementation technology as possible. The elimination of platform specific aspects of application software is essential to achieving portability across a wide range of implementations[6].

3.2 OSE Services

The OSE reference model defines 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 the interfaces specified in section 3.1.

The OSE services are divided into four major categories, which are partitioned into a more detailed set of subcategories:

- System Services
  - Language Services
  - Core System Services

- Communications Services

- Information Services:
  - Database 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 Support Services

3.3 OSE Cross-category Services

The services listed in the previous section are provided by the entities identified in the OSE/RM at their interfaces, allowing access to their associated facilities and services. There is, however, another class of services that may influence the OSE system basic entities. These services are referred to as POSIX OSE cross-category services.

A cross-category service, when applied, may have a direct effect on the operation of one or more of the open system components, but it is not in and of itself a stand-alone OSE component.

Examples of POSIX OSE cross-category services include internationalization, security, and administration.

3.3.1 OSE Internationalization Services

The task of internationalization is to ensure that the services provided by the POSIX OSE, and the interfaces between such services, are specified in such a way that they can be easily used all over the world, supporting multiple natural languages and the underlying cultural conventions.

3.3.2 OSE Security Services

Security services are provided to support the secure distribution and integrity of information, and to protect the computing infrastructure from unauthorized access.

The four basic security objectives of a system are to maintain:

- Confidentiality
- Integrity
- Availability
- Accountability

3.3.3 OSE System Management Services

Information systems are composed of a wide variety of resources that must be managed effectively to be successfully used. Although resources may differ widely, the basic management concepts may be applied in a uniform manner. The adoption of a common management model enables administrators to manage different resources without the need to learn different techniques for each resource.

The goal of system and network management standards is to enable the provision of portable and interoperable management technology[5]. The existence of standard user-level interfaces to management functionality also provides portability of system and network administrators.

Systems management functionality may be divided into functional groups, according to the resources being managed (e.g. printer management, software management, user management, network management, processor management) or into categories, identifying components common to all functional groups:

- Configuration Management
- Performance Management
- Fault Management
- Accounting Management
- Security Management

The reference model for systems management (see figure 6)[5] is consistent with the POSIX OSE model shown in figure 5.

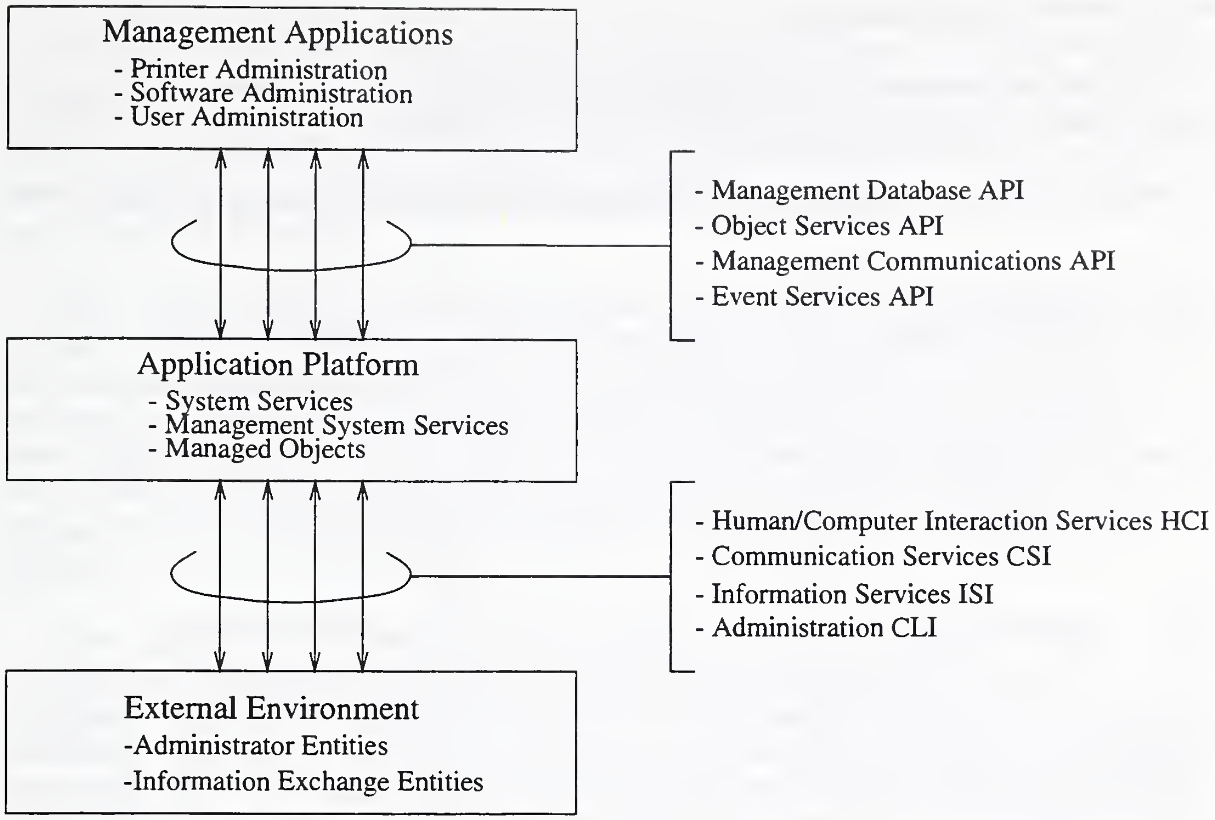
Management Applications
- Printer Administration
- Software Administration
- User Administration
- Management Database API
- Object Services API
- Management Communications API
- Event Services API
Application Platform
- System Services
- Management System Services
- Managed Objects
- Human/Computer Interaction Services HCI
- Communication Services CSI
- Information Services ISI
- Administration CLI
External Environment
-Administrator Entities
-Information Exchange Entities

Figure 6 POSIX OSE Reference Model for Systems Management.

Management applications provide software application entities to support management tasks in specific areas of management functionality. They make use of services provided by the application platform.

In addition to the normal functionality platform services, there are management components within the platform representing management system services. Managed objects are included in the application platform. They are abstract representations of resources that are to be managed. This abstraction allows different resources to be managed in a uniform manner.

The external environment, as described for the systems management reference model, includes administrators who use management applications via a command line interface or a graphical interface facilitating administrator portability, and mechanisms for exchanging information outside the model (e.g. via external networks and transportable media).

POSIX OSE standards for some of the management services -- print, software distribution and user group management -- already exist. Other standards and emerging standards developed by other organizations will be included in POSIX OSE.

4 ODP-OSE Relationships

The ODP OSE reference models address complementary problems. The RM-ODP focuses on enabling distributed systems and the OSE/RM on enabling application portability.

OSE specifications are limited to application platform boundaries, as shown in figure 7.

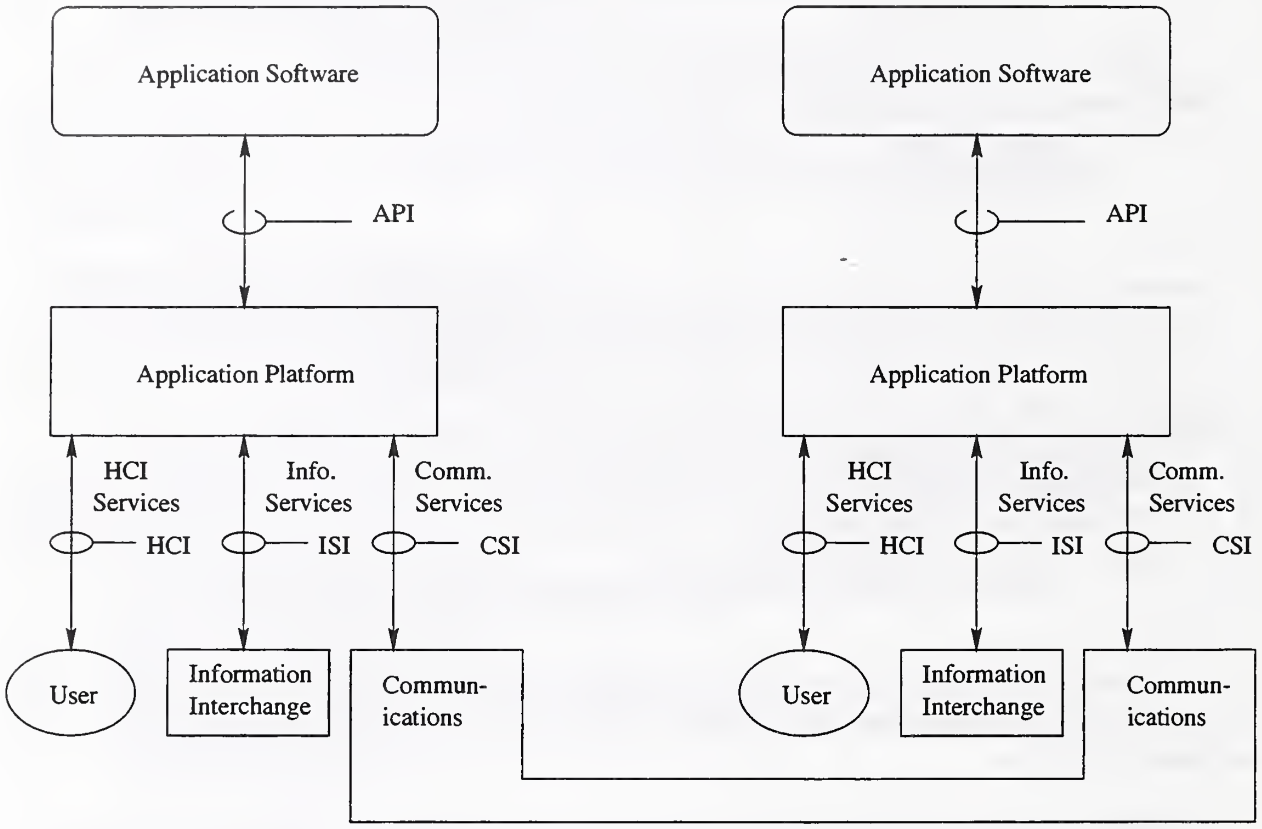

Figure 7 POSIX OSE-based Distributed System

The OSE/RM identifies system entities and interfaces between those entities that are related to application software portability and interoperability. It categorizes and describes services available to application platform users at those interfaces. Therefore, these interfaces characterize the application platform from the application software and external perspective. No further implementation aspects are represented. The application platform is treated as a black box. In a distributed system, the application platform may be implemented as either distributed or non-distributed and the OSE/RM does not distinguish between the two cases.

The elimination of platform specific aspects of application software is essential to achieve the main objective of OSE -- application portability across a wide range of platform implementations.

A programmer wishing to write portable application software in an OSE will use standardized programming languages and language bindings to the required services. A consistent interface to the operating system is also essential for applications portability. Thus, applications are wrapped in standard interfaces that connect them to the computing environment. They produce and accept standard data formats, and communicate using standardized protocols. The use of standard communication protocols, data interchange formats, and distributed interfaces allows applications to interoperate.

Source code portability is also addressed by the ODP reference model in the computational viewpoint. Standards for portability in ODP systems specify action templates for the actions in which a computational object can participate (i.e. initiate operation invocations, respond to operation invocations, instantiate interface templates, bind interfaces, access and modify its state). The specification of such templates is a matter of language design. A portability standard may represent the permitted actions directly (e.g., as library functions) or indirectly through syntactic structures. ODP specifications, however, take a different view of the application platform.

A new approach to distributed systems is emerging. Application platforms are being broken into parts and distributed across heterogeneous environments. Additionally, what has traditionally been regarded as a single application is also being decomposed into parts, and those parts distributed across a heterogeneous distributed environment at runtime. The ODP reference model addresses and enables this distribution of application software and application platform components across distributed systems. In order to achieve these objectives it is necessary to address interfaces internal both to application platform and to external communications entities[6]. The visibility of internal interfaces is the main difference between the ODP and OSE approach.

Providing an architecture which allows decomposition of both application software and application platform into parts that may be distributed over a heterogeneous environment, implies considering issues such as:

- What modifications to existing platform services are required to support the distribution of application software components?
- What additional services are needed?
- How can application platform services be distributed?
- What is the application software implication on the distribution of specific services and interfaces?

These issues are important to both ODP and OSE. Although OSE does not address application platform internal services and interfaces, the provision of those services that support application portability and interoperability are necessary at application platform boundaries.

Similar to the OSE specifications limitation to application platform boundaries, the ODP computational viewpoint is concerned only with the functional decomposition of the system into objects which interact at interfaces without considering distribution. Only the engineering language prescribes the internal structure of the application platform. For both OSE specifications and the ODP computational viewpoint, the application platform view is at its external interfaces. Neither Model considers the internal interfaces. Therefore, the abstraction represented by the OSE reference model appears to correspond to the computational viewpoint of the ODP reference model. A correspondence may be established between components of the two models. Some examples are shown in the following table.

OSE	ODP
Application Program Interface	Computational Interface
Application Software Entities	Basic Computational Objects
Application Platform	Infrastructure described by the Engineering Language

Table 1. Component Correspondence

4.1 Distributed Systems Management

Distributed systems management can be regarded as a particular case of a distributed application. POSIX OSE presents a reference model for systems management (see figure 6) consistent with the OSE/RM. Application entities are in this case management applications that make use of services provided by the application platform. These services are both normal functionality services and additional management services.

The application platform includes managed objects to represent resources to be managed, but the OSE/RM makes no reference to how those managed objects are organized and how management services are provided at the application platform boundaries.

The ODP reference model describes distributed system management concepts, such as managed objects representing both application software components and application platform resources, and management interfaces to support management operations that can be performed on managed objects.

Management support is addressed both in the computational and engineering viewpoints. For example, a binding object in a computational binding may have a control interface that provides the means to express monitoring, and configuration and quality of service control. Management of a computational binding would correspond in the engineering viewpoint to channel management, via control interfaces of stubs, binders, protocol objects and interceptors (see section 2.7).

ODP specifications do not include a management model. They only describe basic management concepts and how the ODP reference model may support different management functional areas. However, a reference is made to some examples for system management[1], namely an example based on the OSI Management Framework and a Distributed Object Manager (DOM) which allows users to identify and access available services in a dynamically configurable distributed environment.

5 References

1. ISO/IEC 10746-1. Open Distributed Processing - Reference Model - Part 1: Overview. ISO/IEC 10746-1, ITU-T X.901 (committee draft), July 1994.

2. ISO-10746-3. Open Distributed Processing - Reference Model - Part 3: Architecture. ISO/IEC 10746-3, ITU-T X.903 (committee draft), 1995.

3. ISO-10746-2. Open Distributed Processing - Reference Model - Part 2: Foundations. International Standard 10746-2, ITU-T X.902, 1994.

4. Application Portability Profiles (APP) - The U.S. Government's Open System Environment Profile Version 3.0. Draft, Computer Systems Laboratory, NIST, 1995.

5. ISO/IEC TR 14252 and IEEE 1003.0. Guide to the POSIX Open System Environment, 1995.

6. *Programme of Research on Open Systems Testing - Report of the Study on Testing for Open Distributed Processing*, Chapter on Conformance Testing, ODP and OSE. U.K. Department of Trade and Industry (DTI) with Architecture Projects Management Ltd., NIST, National Physical Laboratory, Object Management Group, and University of Kent at Canterbury, 1993.