Chapter 10

Interface Design

10.1 INTRODUCTION

Interfaces are common failure points on systems. An interface is a connection resource for hooking to another system's interface (an external interface) or for hooking one system's component to another (an internal interface). The systems engineer's design problem includes identifying the interfaces, both external and internal, and allocating items (inputs and outputs) to the defined interfaces. Once these tasks are completed, the requirements for each interface must be derived from existing system-level requirements. Finally, alternative interface architecture alternatives must be examined, including the needed functions and the most cost-effective alternative chosen.

The interface requirements must address total system performance, the fidelity of the interface, and any system requirements meant to constrain interface design. Typical system performance requirements of concern in designing the interfaces are system throughput and response time. The fidelity of an interface is determined by the integrity of the items being transported, the guaranteed delivery of the items, and failure detection and recovery within the interface. In other words the interface should not change the items during the transmission process, should eventually deliver every item placed on the interface (and not create any items), and should detect faults early and recover gracefully (a hard but important word to define).

Section 10.2 discusses the process for developing the interface designs of the system. Generic architectures, introduced in Section 10.3, can be used as the architectural concept for any given interface. These generic architectures come from communication and computer systems. Section 10.4 discusses the important issue of standards, a major support in the definition and design of interfaces. Sections 10.5 and 10.6 address two major standards, one for communications systems and one for software architectures. The open systems interconnection (OSI) reference model serves as the basis for many standards related to telecommunications and computer networks. This reference model provides a rich basis for viewing interfaces. The common object request broker architecture (CORBA) is an industry standard for software systems integration. Section 10.7 addresses the design of an interface.

The generic interface architectures described in this chapter include message passing, shared memory, and network. Each of these architectures is described, followed by a discussion of strengths and weaknesses.

The OSI reference model and CORBA are introduced as well-conceived architectures for common interfaces. The discussion in this chapter is focused on the functions performed in these architectures so that the engineer of a system has samples of functions to draw from for designing any type of interface.

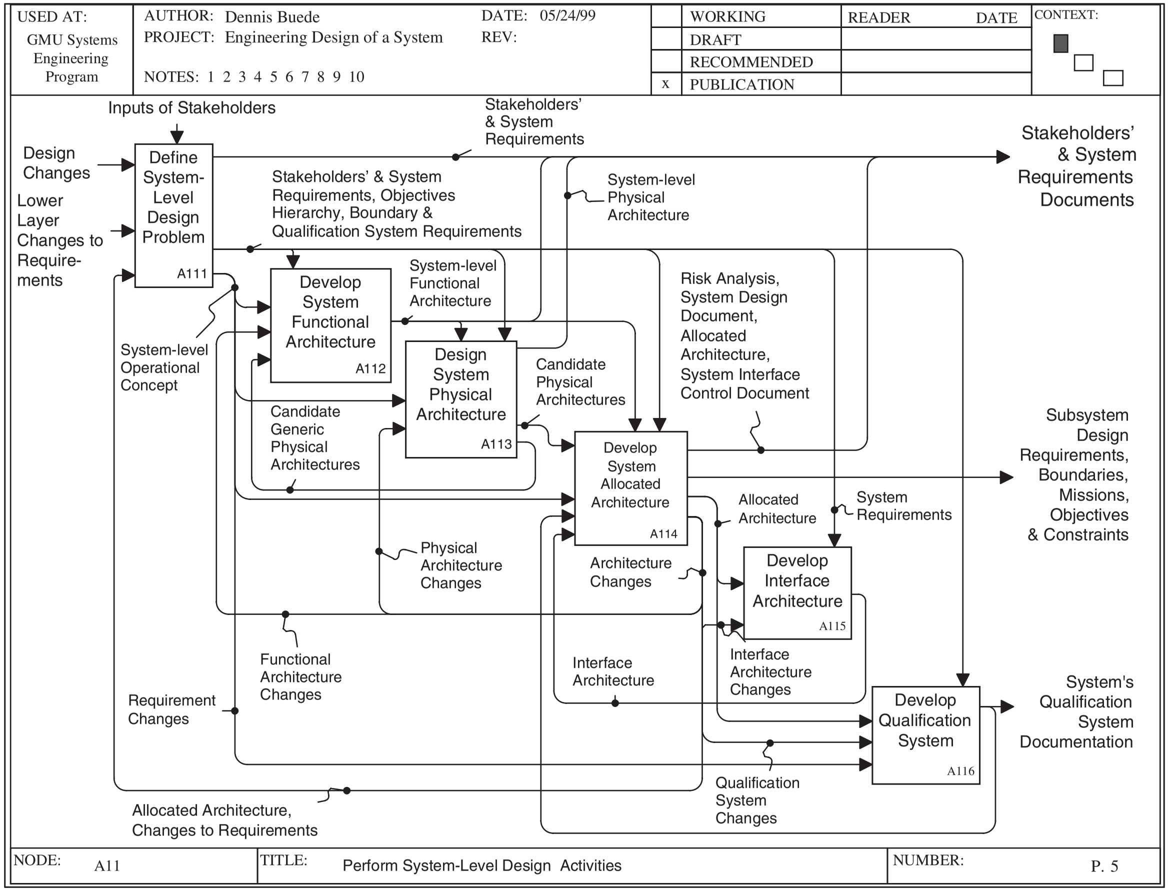
The *exit criterion* for completing the design of the system's interfaces is acceptance by the engineer responsible for the allocated architecture that the interface is consistent with the system's components and configuration items (CIs) as well as the performance objectives and requirements of the system.

10.2 OVERVIEW TO INTERFACE DEVELOPMENT

An *interface* is a connection for hooking to another system (an external interface) or for hooking one system component to another (an internal interface). The interface of a system contains both a logical element and a physical element (or link) that are responsible for carrying items (electro-mechanical energy or information) from one component or system to another. The interface must ensure that the item is delivered on time and in the same form as the item was received.

The development of the interface architecture is quite similar to the development of the allocated architecture of a system, as shown in Figure 10.1. [See Appendix B for the entire IDEF0 (Integrated Definition for Function Modeling) model for engineering a system.] The functions of defining requirements as well as the functional, physical, and allocated architectures are present. The only new function is the evaluation and selection of a high-level interface architecture; Section 10.3 defines and discusses the three major alternate interface architectures in use today in communication and computer systems. This high-level architecture for the interface is analogous to the concept selection for the system design. Before proceeding very far in the development of a system, high-level concepts, each having a different operational concept, are posited and evaluated.

This decomposition of functions for developing an interface architecture assumes that the functional process will be revisited several times in whole or in part. As interface changes arrive from the process responsible for the system's allocated architecture, the relevant functions for developing the interface


USED AT:
GMU Systems Engineering Program
AUTHOR: Dennis Buede
PROJECT: Engineering Design of a System
NOTES: 1 2 3 4 5 6 7 8 9 10
DATE: 05/24/99
REV:
WORKING
DRAFT
RECOMMENDED
x PUBLICATION
READER
DATE
CONTEXT:
Inputs of Stakeholders
Design Changes
Lower Layer Changes to Require-ments
Define System-Level Design Problem
A111
Stakeholders' & System Requirements
Stakeholders' & System Requirements Documents
Stakeholders' & System Requirements, Objectives Hierarchy, Boundary & Qualification System Requirements
System-level Physical Architecture
Develop System Functional Architecture
A112
System-level Functional Architecture
Design System Physical Architecture
A113
Candidate Physical Architectures
Risk Analysis, System Design Document, Allocated Architecture, System Interface Control Document
System-level Operational Concept
Candidate Generic Physical Architectures
Develop System Allocated Architecture
A114
Subsystem Design Requirements, Boundaries, Missions, Objectives & Constraints
Allocated Architecture
System Requirements
Physical Architecture Changes
Architecture Changes
Develop Interface Architecture
A115
Functional Architecture Changes
Interface Architecture
Interface Architecture Changes
Requirement Changes
Develop Qualification System
A116
System's Qualification System Documentation
Qualification System Changes
Allocated Architecture, Changes to Requirements
NODE: A11
TITLE: Perform System-Level Design Activities
NUMBER:
P. 5
FIGURE 10.1 Development process for the interface architecture.

architecture are triggered and set the whole process in motion to develop a revised interface architecture.

10.3 INTERFACE ARCHITECTURES

Most interfaces are communication systems or analogies of communication systems (e.g., a conveyer belt). *The principal communications architectures are message passing, shared memory, and networks*. An every day example of each of these architectures follows:

Message Passing: mail delivery that predictably occurs once or twice a day and allows those receiving the mail to turn their attention to the mail immediately or wait until a more opportune time and permits messages of substantial volume.

Shared Memory: a meeting or conference in which only one person speaks at a time and conveys relatively compact messages; all can hear what is said but yet are restrained from other productive work during the meeting.

Network: a telephone conversation that can involve messages of widely varying lengths and can be instigated at almost any time.

10.3.1 Message Passing Architectures

The message passing architecture is used to allow the predictable exchange of information. The message passing architecture is commonly found as an internal interface in systems since the systems engineers have the information to determine whether the message is predictable. A message passing architecture can also be found as an external interface among a number of systems that have consistent message traffic.

The physical architecture for message passing typically currently involves up to 32 nodes on a linear, bus topology connecting the nodes. Included in the architecture are the bus interchange unit, transceivers for the nodes, and signal lines.

The message that is transmitted over the bus consists of a protocol and data segments. The protocol segment includes any information needed by the bus interchange unit to deliver the message; typically this is information about the size of the message and address of the node to receive the message.

For each transmitted message the following communication process must be completed:

1. One node must win control of the communication channel by a priority scheme implemented by the system.
2. The winning node becomes the master and sends a protocol segment to the intended receiving node(s), called the slave(s).

3. The slave node(s) notifies the master that the protocol segment was successfully received.
4. The master sends (or receives) the data segment to (from) the slave(s).
5. The slave(s) notifies the master that the data segment transfer is complete.
6. The master surrenders control of the communication channel.

The most common application of message passing is for systems that can define a predictable message transmission schedule upon initialization. Update rates for messages are on the order of 0.01 to 1 second. Other types of message passing can occur (such as asynchronous communication that can be predicted statistically but not predefined) but the message passing architecture is not preferred if these types of messages are substantial portions of the traffic.

10.3.2 Shared Memory Architectures

Asynchronous communication requests of a byte to a few words in size that can be defined statistically are ideal for shared memory architectures. The shared memory architecture is a fast access storage device, typically a memory device, which is the interface among processors. The shared memory and interacting processors can either be part of the same hardware component or interface via global memory. Statistical predictions of message traffic are usually possible when message updates are within several clock cycles (e.g., nanoseconds).

The communication model for shared memory is:

1. A processor generates a read or write request for another address in shared memory.
2. The current owner of this variable is notified of the request.
3. The cache memory of the current owner is dumped to local memory.
4. The global variables of the current owner are dumped to shared memory.
5. The read or write request of the processor is completed with a data transfer.

Performance of shared memory systems can be degraded substantially if a requesting processor needs information that is not in the cache memory of the shared memory interface. In this case all activity is blocked until the shared memory can retrieve the variables needed. Shared memory works best in highly parallel software applications in which the global data of each application must be accessed frequently by the application and infrequently or never by the other applications.

10.3.3 Network Architectures

Networks have become commonplace in the workplace with the local area network (LAN) products. In many ways the network architecture is a distributed collection of shared memory systems, in which each shared memory system has the ability to tap into the shared memory of the other systems on the network. The best analogy for communication is to a file server with access to slow storage devices; in this case the communication of information is via a statistical block transfer process.

The transfer of information typically takes milliseconds to minutes, depending upon the size of data set, and includes relatively large blocks of data. The main difference between the network and the message passing architecture is that a network provides demand-based service while message passing primarily uses scheduled transfers. Networks can service hundreds of nodes, while message passing is currently limited to 32 or fewer.

A network system typically includes the communication hardware and a software package, typically called a network operating system. There are many such commercial network operating systems. The software provides various priority-based queueing models, often with separate transmit and receive queues. The network provides extensive fault checking and does not suffer from the failure modes of message passing architectures.

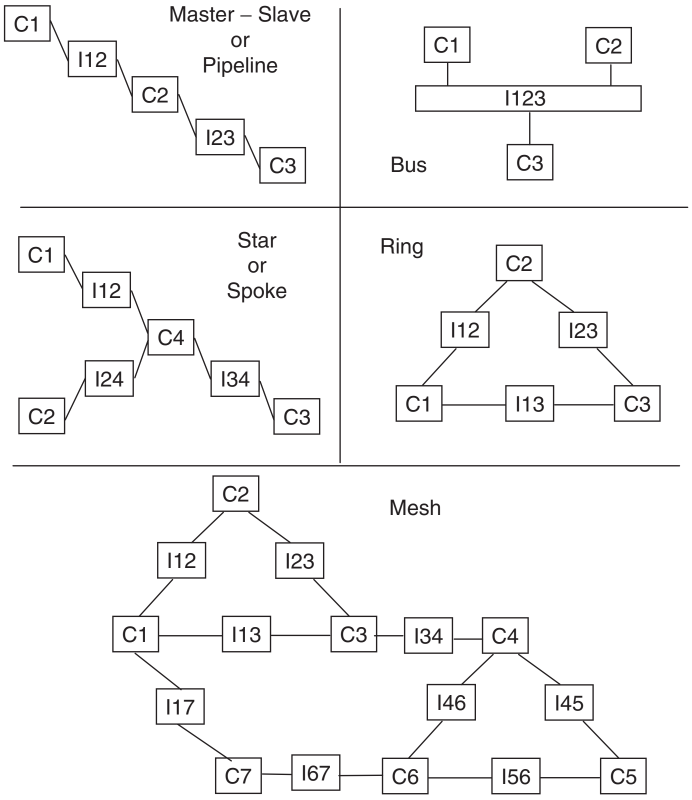


FIGURE 10.2 Network architectures.

There are many network architectures available. Five of the most common are shown in Figure 10.2. The pipeline architecture is a serial linkage of components that is most appropriate when the components only need to communicate with their neighbor in the network. The bus architecture is the most general; each component places its information on the bus, and the bus distributes the information to the appropriate sources. The bus architecture is most appropriate for a large number of components. The spoke architecture isolates one component as the central processor that manages the communication process. The ring architecture is one of the most common architectures in office settings. The mesh architecture is an irregular connection of components that provides sufficient redundancy (pathways between any two nodes) for the system under consideration while stopping short of full interconnection. Duato et al. [1997] provides many examples of interconnection networks used within parallel computation devices and telephony systems.

10.4 STANDARDS

Standards help ensure that an interface will enable the connection of two components. Each component is required to meet a given standard, and the interface is designed to meet the same standard. As long as the performance associated with the interface and the associated standard are satisfactory, the design will be successful.

Standards have different levels of formality: formal, de jure, and de facto. Formal standards are negotiated and promulgated by accredited standards bodies, such as the International Organization for Standards (ISO), International Telecommunications Union (ITU), and the American National Standards Institute (ANSI). Professional societies also develop and promulgate standards. Examples of such professional societies are the Institute of Electrical and Electronic Engineers (IEEE) and the Electronics Industry of America (EIA).

Legal authorities mandate de jure standards. For example, the IDEF0 standard is a federal information processing standard (FIPS) that was created by the National Institute of Standards and Technology (NIST) of the U.S. government.

De facto standards come into existence without any formal process. Popular usage creates de facto standards. X Windows and the Windows operating system are examples of de facto standards.

The benefits normally attributed to using standards are interchangeability, interoperability, portability, reduced cost and risk, and increased life cycle. Interchangeability is the ability to interchange components with different performance and cost characteristics. In this way creating multiple versions of a system in which one or more components are interchanged is possible because the adoption of these standards makes the interchange possible. Most computer manufacturers have adopted sufficient standards so that they create multiple versions of a specific design with varying central processing unit (CPU) performance speeds, varying amounts of random-access memory (RAM), and varying size hard discs for storage.

Interoperability benefits of adopting standards accrue because the system can now operate with a wider variety of external systems, systems that have also adopted the same conventions. For example, computer manufacturers that adopt the standard parallel and serial interfaces can be interfaced with a wide variety of peripherals such as printers. The benefits for most systems to be interoperable with other systems are so great when standards exist that it is difficult for system designers to deviate from such standards. The answer for such deviations is limited performance by an aging technology. Predicting if and when a new technology will provide enough increased performance or decreased cost to justify changing a standard is often difficult.

Portability is a benefit for systems that operate on another system. Software systems obtain portability by adopting the standards necessary to run on multiple platforms with varying hardware or operating systems. Systems that require power obtain portability by having a power unit that permits power to be obtained from a standard wall socket. Systems like my laptop computer that require direct current (dc) current still need the portability to operate using power from alternating current (ac) sources and include a power unit that converts ac to dc power.

Adopting certain standards allows a system designer to buy modules that provide the needed performance characteristics at reduced cost. Standards promote competition among vendors, competition that provides reduced cost and reduced risk for equivalent performance.

An increased life cycle for the system is possible when long-lived standards are adopted. The system can use the interoperability of its components to upgrade its capabilities as new technologies come along, as long as these new technologies adopt the standards. Typically the new technologies provide downward compatibility in the sense that the older products can be replaced by the new, but not vice versa.

10.5 OPEN SYSTEMS INTERCONNECTION ARCHITECTURE

In 1977 the ISO approved the initiation of work on a standard for the interconnection of computers comprised of different architectures and technologies [MacKinnon et al., 1990]. The first meeting, involving 40 experts, was held in March 1978. At the time a number of proprietary communications architectures were available (e.g., Digital Network Architecture (DNA) of Digital Equipment Corporation, Distributed Systems Architecture of Honeywell, and Systems Network Architecture (SNA) of IBM). In 1983 the ISO and the International Telephone and Telegraph Consultative Committee (CCITT) of the ITU approved the reference model for OSI [Schwartz, 1987]. This reference model defines a seven-layer architecture for network-based communication between end-user nodes in a telecommunications network. The OS} is a set of internationally accepted standards that revolve around this reference model; these standards were developed in international forums and have been accepted on an international basis for this reason. The OS} is also a set of products that conform to these standards.

The OSI reference model contains seven layers: physical, data link, network, transport, session, presentation, and application. The first four layers are known as the lower network layers. The last three layers are known as the higher layers; these higher layers plus the first four layers must be present in each end user or host node. On the other hand, intermediate nodes in the communications architecture must only possess the first three layers. Figure 10.3 presents a common representation of communication between two hosts using a communications network, such as a LAN or the Internet. Data is being transferred from an application on the left host node through the physical media and an intermediate node in the communications network to the host node on the right. The number of intermediate hosts depends not only on the communication network but on the route selected through that communication network. In the communication network at the top of Figure 10.3 at least two intermediate nodes would be involved in communication between the two hosts shown; it is possible that all five nodes would be involved.

Some of the key definitions associated with OSI are [MacKinnon et al., 1990]:

System: an autonomous whole capable of performing information processing or information transfer.

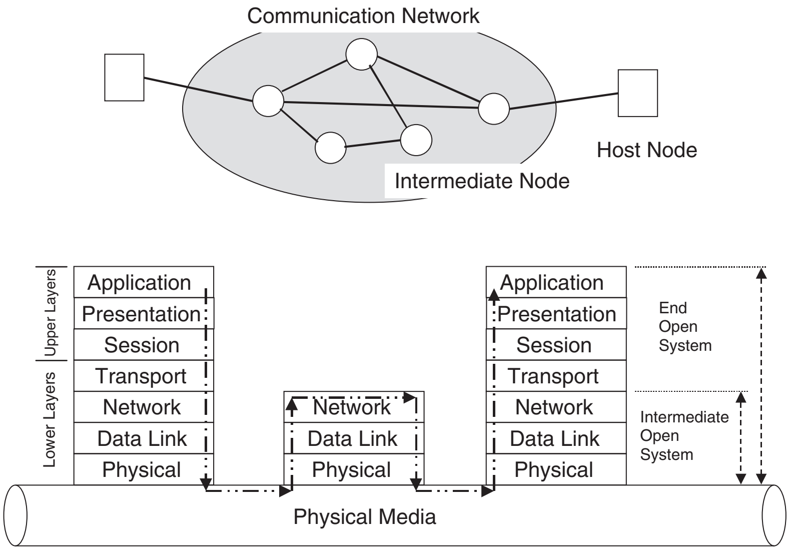


FIGURE 10.3 Communication in the OSI reference model.

Open System: a system than can create, transmit, receive and act upon OSI messages.

Interconnection: ability to satisfy four types of activity — movement of digitized data over physical transmission media in a reliable manner; organization and control of the paths between those open systems that are the sources and destinations of information; exchange of commands and data to manage the cooperation of the systems that desire to interwork to achieve a specified purpose; and provision of a variety of services and facilities that directly support the user applications.

Service Provider: the subsystem formed by a layer and all layers below it. This subsystem only serves the layer above it. So the service provider formed by the transport layer includes the network, data link, and physical layers and serves the session layer.

Protocol: a complex multipart message that is passed between systems.

Protocol control information (P-N): information that is added at layer N to the front of a message received from the (N + 1) layer above; this information is used to control the transmission of the message among entities in layer N.

Protocol data unit (N-PDU): the message at layer N that contains the message from layer N 4-1 plus the protocol-control-information for layer N.

Interface control information (I-N): information that is added at layer N to the front (and possibly the end) of the protocol-data-unit of layer N to be sent to layer N−1.

Interface data unit (N-IDU): the message at layer N that contains the interface control information plus the protocol data unit of layer N and that will be sent to layer N−1 for transmission on the N−1 service provider.

Service access point [(N)-SAP]: the point of interaction between layers N + 1 and N; the point at which I-N is added to the front (and possibly end) of the N + 1-PDU being sent from layer N + 1.

Application: a set of distributed tasks that satisfy some real-world information processing requirement.

Application entity (AE): the portion of an application that is responsible for interconnecting via OSI.

Presentation entity (PE): the presentation protocol functionality within an open system that transforms data syntax so that the data can be transferred properly.

(N)-entity: the functionality within layer N that adds P-N as one of its functions.

Subnetwork: a real communication network.

First, note the narrowness of the definition of system chosen in this domain. Second, the multilayered model of a communication system both enables an orderly development of standard products and creates a significant overhead for communicating information.

Figure 10.4 illustrates the process of moving data from one application to another over an OSI-compliant network and the overhead associated with that movement. The adding and stripping of information at each of the seven levels is necessary to make this movement happen but increases the data size. As data enters the OSI-compliant product at the application service access point [(7)-SAP], nI-7 information is added to the front end of the data. This augmented data is then received at an (AE), where P-7 information is added to the front end, forming the 7-PDU. An imaginary transfer of the 7-PDU takes place on the presentation service provider (indicated by the dashed horizontal line in the application layer). In reality the 7-PDU is sent to the Presentation layer where 1-6 is added at the (6)-SAP and P-7 is added at the PE, forming the 6-PDU. This process continues through the first layer where the 1-PDU is actually placed on the physical media and transferred to the correct host. The process is repeated in reverse with the protocol and interface-control-information being stripped at successively higher layers until the original data is delivered to the application on the second host.

Table 10.1 provides a short description and the key functions of each layer [Levi and Agrawala, 1994; MacKinnon et al., 1990; Schwartz, 1987]. Each

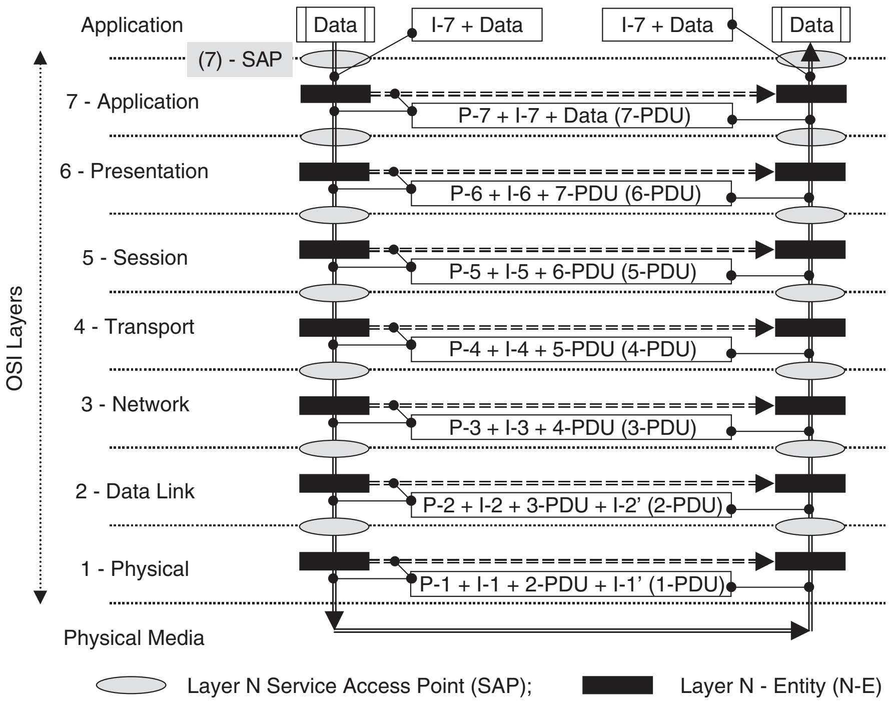


FIGURE 10.4 OSI process of adding and stripping PCIs and ICIs.

TABLE 10.1 Summary of OSI Reference Model

Layer	Description of Layer	Layer Functions
(7) Application	Provides necessary communications between the end user's application processes and the application-entity. The application-entity is the key operator of this layer. The two primary modes of communication are connection and connectionless. (The following discussion in this table addresses the connection mode.)	• Establish connection (receive request, send indication, receive response, send confirmation) • Transfer data (receive request, send indication, receive data, initialize data, associate data, send data) • Release connection (receive request, send indication)
(6) Presentation	Defines data syntax for communication between application-entities and maintains transparency to the hosts. The presentation-entity is the key operator of this layer.	• Establish connection • Transfer data (receive request, send indication, negotiate syntax, receive data, transform syntax, send data) • Release connection
(5) Session	Provides connection control for the hosts by enabling presentation-entities to organize the exchange of data in either full or half-duplex mode.	• Establish connection • Transfer data • Establish synchronization points • Manage activity • Release connection • Report exceptional conditions
(4) Transport	Establishes transparent and reliable end-to-end transmission of data between host nodes.	• Establish connection • Transfer data • Provide error detection and recovery • Release connection
(3) Network	Determines the establishment of connection without concern for the type of sub-network and handles routing. Represents the interface between the communications carriers (layers 1-3) and the computer manufacturers (layers 4-7).	• Establish connection • Transfer data • Perform multiplexing • Provide error control • Provide sequencing and flow control • Release connection

(*Continued*)

TABLE 10.1. Continued

Layer	Description of Layer	Layer Functions
(2) Data Link	Establishes reliable transmission on the physical layer.	• Establish connection • Negotiate quality of service (QOS) • Transfer data • Provide flow control • Reset connection • Release connection
(1) Physical	Defines how the physical network is accessed in order to provide bit transparent transmission on the physical media. Supports synchronous and asynchronous transmission; duplex, half-duplex, and simplex modes; and point-to-point and multi-point topologies.	• Determine presence of signaling pulses. • Determine timing of signaling pulses

layer, except the first, is responsible for establishing a connection on the service provider below it, transferring the data to and from that service provider, and releasing the connection when finished or required. In addition, the layers conduct functions such as reporting exceptional conditions, providing error control, negotiating quality of service, and providing flow control.

While the OSI reference model has received a lot of attention as a standard, the world of products that incorporate communications systems has largely passed OSI by in favor of the de jure standard codified by the military: Transport Control Protocol/Internet Protocol (TCP/IP). This de jure standard has three layers above the physical layer: the network layer for which the LP is defined, the transport layer for which the TCP is defined, and the upper layers, which employ a variety of protocols.

10.6 COMMON OBJECT REQUEST BROKER ARCHITECTURE

From the inception of software applications, one of the most difficult problems for users is the communication of information among software applications developed by different organizations or programmers. Most software applications were designed to be a closed system, often involving proprietary code, algorithms, and interfaces. On occasion, several software applications were integrated vertically to address the problems in a single market. The Object Management Group (OMG) began operations in 1989 in response to this problem. The result is the common object request broker architecture (CORBA) as a standard that would permit programmers to integrate software modules resident on the same network by treating each application as an object. The CORBA standard was developed via a set of request for proposals developed by the OMG and subsequent development contracts issued to corporations such as Digital, HP, HyperDesk, and Sun.

The CORBA standard is actually all three standard types: formal, de jure, and de facto. Part of CORBA, the interface definition language (IDL), is a formal standard that has been adopted by the ISO and the European Computer Manufacturers Association (ECMA). The CORBA is a de jure standard in the United States and among several contractors and a de facto standard elsewhere in the world. The OMG and X/Open jointly publish CORBA.

The CORBA standard treats software applications as objects, and as such, sits at the application level of OSI's seven-layer architecture. See Figure 10.3. The CORBA is based on a client–server model for distributed computing. The IDL, a formal standard, is a universal notation for software interfaces defining a boundary between the client code (requests for services) and the software objects that implement those services. These software objects may be written to the standards defined by CORBA or may be legacy software that is "wrapped" by additional code that does adhere to CORBA standards. The IDL is both platform and language independent and has not changed significantly since first defined in 1991. In fact, IDL must remain stable or the associated standards inherent in CORBA will be broken. The IDL standard defines what is exposed in the interface between the service and its client(s); any other details and relationships are forbidden. For details on the IDL see Mowbray and Ruh [1997] or Mowbray and Zahavi [1995].

Although IDL is the key to making CORBA work from both a software development and architecture perspective, there are four additional categories of objects that comprise the CORBA architecture and are more important to this discussion of interfaces: the object request broker, CORBAservices, CORBAfacilities, and CORBAdomains.

The first object category is the object request broker (ORB), which is the core of CORBA and is an analogy to a bus network. The ORB is the interface between the client (software package requesting a service of another package) and the server (software package performing the service requested). So, in fact, the ORB can be viewed (Fig. 10.5) as a bus architecture that operates in the application layer of the OSI network communication model. The main role of the ORB is to standardize access between software applications, enabling CORBA to hide the programming, platform, and location peculiarities of client and server software objects. Each software object registers its interface characteristics with the ORB. The ORB receives all requests for service by another software application and knows which application to task with the request, where that application is, and how the request has to be translated so that the application will understand the request. The ORB requires that each software application be written in accordance with CORBA standards as

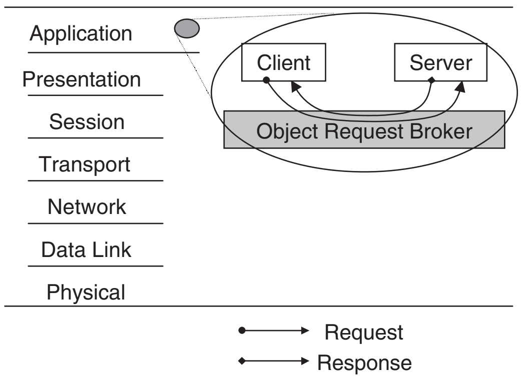


FIGURE 10.5 CORBA overlaid on OSI seven-layer model.

defined by the IDL or wrapped in a software application (wrapper) that adheres to IDL and interfaces with the non-IDL software application. This bus architecture is the reason that CORBA can be efficient in interfacing software applications. Without an ORB-like network each application must be able to interface with every other application; if there were *N* applications and a new one is added, the new application must have *N* new interfaces developed. With CORBA each new application requires either an IDL wrapper to connect it to the ORB or the adherence to the IDL architecture.

Parts of the ORB are exposed to the applications (clients and servers), as shown in Figure 10.6. The dynamic invocation, the ORB interface, and the dynamic skeleton are defined as part of the CORBA specification and provided by all ORB environments. The ORB interface contains several general purpose methods.

The dynamic invocation interface allows the client to request a service without requiring that precompiled stubs be part of the ORB. Dynamic

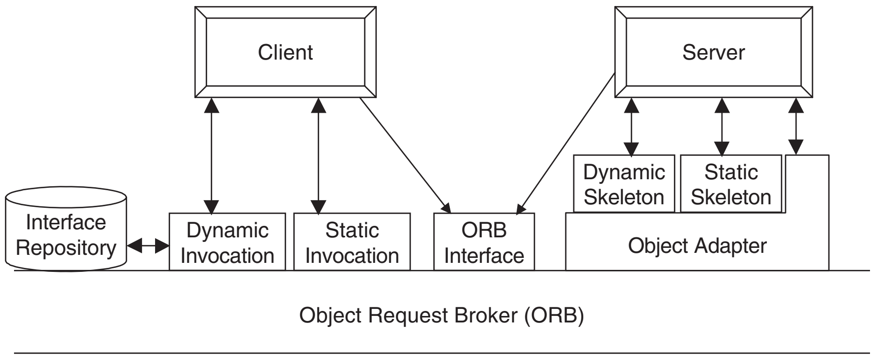


FIGURE 10.6 ORB interactions with clients and servers.

invocation means that interface-related information about the server is acquired at the time of the invocation, providing great freedom and flexibility. The dynamic skeleton associated with the server's interaction with the ORB provides a dynamic bundling of the information in the request from the client into input parameters for the server and a dynamic bundling of the results obtained by the server for return to the client. The combination of dynamic invocation and dynamic skeletons enable users to create implementations of objects that form a gateway to often-used applications such as word processing and databases.

Static invocations (sometimes called stubs) and static skeletons are also available as extensions of the ORB. A static invocation is precompiled on the basis of the IDL interface of the client to the ORB and requires that the client have knowledge of server's characteristics before the request is made. As additional objects (software applications) are added to the ORB, a client relying on static invocation will have to be updated in order to access the new applications. A client using dynamic invocation will be able to learn the needed information from the interface repository while building the request. Interestingly CORBA is constructed so that the server is unaware of the nature (static versus dynamic) of the invocation. (The word "common" was added to CORBA when the decision was made to implement both static and dynamic invocations.) The static skeleton is analogous to the static invocation but on the server side. Static invocations and skeletons have the benefits of being easier to program, performing faster (dynamic invocations can be up to 40 times slower than static invocations [Orfali et al., 1997, p. 71], more robust, and easier to understand.

The final part of the ORB that interacts with servers is the object adapter. The major function of the object adapter is to define how an object is activated. One software application that can satisfy many types of requests could use a different object adapter for each request type. The CORBA standard requires that a basic object adapter be available in every ORB; this basic object adapter is sufficient for most applications. The basic object adapter performs the following functions: installation and registration of an object implementation (implementation repository), generation and interpretation of object references, activating and deactivating object implementations, invoking methods and passing method parameters.

CORBAservices include the types of services that are part of operating systems and are globally applicable. These services are packaged as objects with IDL interfaces and are augmentations of the ORB. Table 10.2 describes the services that currently comprise the ORB-object service (ORBOS) architecture. Additional services are planned for the future. These services enhance the effectiveness, efficiency, and security of the ORB and were proposed by platform and ORB vendors. Each service is implemented as an object so that it can be used by any application.

CORBAfacilities are objects that provide services to application objects and are keyed to interoperability issues of the applications. The initial architecture

TABLE 10.2 Services in the ORBOS Architecture

ORBOS Segment	Service	Description of Service
Information Management Services	Properties	Associates named values or properties with an object.
	Relationship	Creates and provides mechanisms to traverse dynamic links between objects.
	Query	Provides a superset of the structured query language (SQL) queries, based on SQL3.
	Externalization	Processes data structures and object states into flat representations so that the information can be transmitted in and out of objects as a stream.
	Persistent Object	Provides a protocol for a persistent object to store its state in an object database, relational database, or file.
	Collection	Generically creates and manipulates common collections of objects.
Task Management Services	Events	Passes event information among sources and consumers; information can be multicast to registered objects.
	Concurrency	Provides a lock management structure based on either transactions or threads; includes read, write, upgrade, intention read, and intention write locks.
	Transactions	Enables the manipulation of the state of multiple objects for flat and nested manipulations.
System Management Services	Naming	Enables objects to locate other objects by name, and to bind and resolve to directories, analogous to the "white pages of the phone book."
	Lifecycle	Enables the creation, copying, moving and deleting of objects on the ORB.
	Licensing	Allocates objects based upon the number of licenses obtained from the publisher.
	Trader	Enables objects to publicize their services and bid for jobs; analogous to the "Yellow Pages."
Infrastructure Services and Elements	Time	Synchronizes time in a distributed object environment.
	Security	Supports authentication, access control lists, confidentiality and non-repudiation; manages the delegation of credentials between objects.
	Messaging	Enables asynchronous invocations on the ORB.

for CORBAfacilities is divided into user interface management, information management, system management, and task management. Note these are the same elements as CORBAservices except that user interface management in CORBAfacilities replaces infrastructure services and elements in CORBAservices. The applications in CORBAfacilities are likely to change the way the user views computing and to enable the ORB to distribute the computing associated with a user's need across the platforms associated with the ORB in the most efficient manner.

CORBAdomains is the final category of the CORBA. This category is still under development and will facilitate vertical application development in domains such as banking, manufacturing, multimedia conferences, telecommunications, and medicine.

CORBA is not unique in its efforts to enable integration of software applications for users. Other attempts to integrate applications are the distributed computing environment (DCE) of the Open Software Foundation (OSF) and Microsoft's distributed component object model (DCOM). In fact, these three approaches compete with and complement each other. The remote method invocation of JAVA is also related to these three approaches. See Mowbray and Ruh [1997] for a comparison of these approaches.

10.7 INTERFACE DESIGN PROCESS

Interface design is central to the success of the systems engineering process. By determining what the system's components are and allocating functions to these components *in* the process of defining the allocated architecture. Engineers of the system identify those items (inputs and outputs) that pass between components. The transportation of these items must be allocated to some physical entity; additional low-level functions must be defined that make the transition across this transportation entity possible. The IDEF0 diagram in Figure 10.1 shows the design process of the system-level interfaces. As discussed earlier, this design process has all of the elements of the system's design process.

Design of the interface must pay special attention to the system performance issues associated with the interfaces outputs. Concerns about the timeliness, accuracy, and reliability of the outputs of the interface need to be considered carefully. The fidelity of the interface is defined as the insurance of the integrity and delivery of items being transferred; that is, the item being sent is the same as the item being delivered, and the item is delivered in a reasonable amount of time. Clearly the interface needs to be sized to handle some determinable quantity of items. Finally, there must usually be extensive failure detection and recovery algorithms to address the integrity and delivery of items.

The design process for an interface includes the steps shown in Figure 10.7. First, defining the components to be addressed, the items that are transferred between them, and any interfaces that have already been specified should bound the interface design problem. Next, we must identify those items that are

- Define Interface Requirements
  - Identify the Items to Be Transported by the Interface
  - Define the Operational Concept
  - Bound the Problem with an External Systems Diagram
  - Define the Objectives Hierarchy
  - Write the Requirements
- Select a High-Level Interface Architecture
  - Identify Several Candidate Architectures
  - Define Trial Interfaces for Each Candidate
  - Evaluate Alternatives against Requirements
  - Choose High-Level Interface Architecture
- Develop Functional Interface Architecture
  - Specify Functional Decomposition
  - Add Inputs and Outputs
  - Add Fault Detection and Recovery Functions
- Develop Physical Interface Architecture
  - Identify Candidates based upon High-Level Architecture
  - Eliminate Infeasible Candidates
- Develop Allocated Interface Architecture
  - Allocate Functions to Components of the Interface
  - Analyze Behavior and Performance of Alternatives
  - Select Alternative
  - Document Design and Obtain Approval

FIGURE 10.7 Interface design process.

to be included in the interface for which we are concerned. Before getting into the design, we must derive the requirements for this interface from the current requirements specification. Included in these requirements are the performance, cost, and trade-off requirement that will be instrumental in selecting the interface.

The design steps are to choose an interface architecture (e.g., shared memory, message passing, bus network); define specific trial interface alternatives (e.g., various bus network alternatives); evaluate these alternatives against the requirements, specifically the performance, cost, and trade-off requirements; and finally choose a specific interface alternative.

Once the interface has been chosen, the behavior of the interface must be detailed and added to the functional architecture. Next, functional behavior is allocated to the existing components and the new interface. Finally, the performance of the segment containing the components and interface must be evaluated, and critical fault detection and recovery behavior must be added to the functional architecture and then allocated to the components and interface.

Figure 10.8 provides a sample result of the above interface design for an elevator. The interface is an external one between the elevator and the building for the purpose of transferring emergency communications between passengers in the elevator and appropriate emergency response unit (e.g., police). In this case a standard interface item, a commercial telephone system, is chosen,

- Define Interface Requirements
  - Identify the Items to Be Transported: Emergency Communications from the Elevator to the Building (and onto the emergency response team)
  - Define the Operational Concept: Passenger encounters emergency and requests ability to make emergency known to emergency response team; Elevator provides resource for passenger to use; Passenger communicates
  - Bound the Problem with an External Systems Diagram: (skipped)
  - Define the Objectives Hierarchy: Objectives are (1) availability of interface, (2) fidelity of the communicated message, (3) operational cost (monthly) and (4) deployment cost.
  - Write the Requirements: (skipped)
- Select a High-Level Interface Architecture
  - Identify Several Candidate Architectures: (1) Telephone connection to building, (2) Dedicated communication system network to emergency response team
  - Define Trial Interfaces for Each Candidate (skipped)
  - Evaluate Alternatives against Requirements: Dedicated network is too expensive to install
  - Choose High-Level Interface Architecture: Telephone connection is chosen
- Develop Functional Interface Architecture: Not needed because interface is standard
- Develop Physical Interface Architecture Not needed because interface is standard
- Develop Allocated Interface Architecture Not needed because interface is standard

FIGURE 10.8 Sample interface design between elevator and the building housing the elevator.

making most of the interface design process unnecessary. Commercial standards are often chosen as interfaces for this reason.

10.8 SUMMARY

Interfaces are the primary responsibility of the systems engineer and are the most common failure point on systems. Designing the interfaces of a system begins with identifying the interfaces, both external and internal, and allocating items (inputs and outputs) to the defined interfaces. Next the requirements for each interface must be derived from existing system-level requirements. As part of the system's requirements, interface requirements will be derived that define the performance and fidelity of the interface. System throughput and response time are the common performance issues that are relevant to designing the interfaces. The fidelity of an interface means ensuring the quality of the items being carried.

As part of the design process alternative interface architecture options must be examined and the most cost-effective chosen. These alternatives can be based on message passing, shared memory, or network architectures, depending upon the characteristics of the items being transported and the performance issues associated with the system.

Standards play a major role in the design or selection of an interface. If a standard can be selected as an interface, then the design information that needs to be communicated in any component or CI specification is readily available and probably well understood. Standards can be formal (adopted by a recognized standards-setting body), de jure (mandated by legal authorities), and de facto (adopted via popular usage by many commercial concerns).

Two major standards, the open systems interconnection (OSI) reference model and the common object request broker architecture (CORBA) were presented in this chapter. These two standards demonstrate the complexity associated with most significant interfaces in terms of design issues and functionality.

CASE STUDY: PATHFINDER COMMUNICATIONS FAILURE

The Pathfinder system that was deployed to the surface of Mars for a landing on July 4, 1997 was truly a success in many ways. Unique system design features included a landing on air bags and the small but effective Sojourner rover.

However, a few days into the mission operators on the ground noticed that infrequent total system resets were occurring that were causing the loss of data. The Pathfinder's information system contained an interface described as a "bus or shared memory area" [Jones, 1997]. A priority system had been established for giving various system activities access to this interface. A bus management task had high priority and ran frequently to accept specific data elements into the shared memory area and then distribute them to their proper locations. A task for gathering and publishing meteorological data had low priority. A particular, lengthy communications activity employed by the spacecraft had a medium priority. Mutual exclusion locks were employed to give an activity access to the interface. A mutual exclusion (mutex) lock is given to an activity and grants that activity control of the communications interface until it releases control back to the interface. VxWorks is the commercial package used on Pathfinder to handle these scheduling activities on the interface. Wilner [1997] described the problem causing the system resets and the process used to diagnose and fix this problem.

The meteorological data gathering activity was an infrequent user of the communications interface and involved the publishing of a substantial amount of data. This data was so voluminous that the meteorological data activity would have to obtain and release mutexes several times before it was finished. The meteorological activity was broken into short enough segments that the high-priority bus management task could gain control for its important functions during the meteorological activity. However, the long running, medium-priority communications activity would infrequently interrupt the meteorological activity during one of its pauses and gain control of the interface. The durations of this medium priority communications task and the previous segments of the meteorological task were sufficiently long to invoke a watchdog timer that was employed to ensure that the high-priority bus management task was executing appropriately. In these rare cases the watchdog timer would invoke a total system reset as a hedge against the system being in a deadlock or failure mode. Whenever the reset occurred, the data in the interface would be lost.

Fortunately, VxWorks had a feature for recording a total trace of system events. Jet Propulsion Lab (JPL) engineers ran the Pathfinder replica on Earth in their lab until the reset situation was replicated. They found that VxWorks had been programmed to run without a feature called priority inheritance. Enabling this priority inheritance feature would solve this problem by keeping the medium-priority communications task from slipping into the middle of the meteorological publishing task. The JPL engineers uploaded a short C program that enabled the priority inheritance feature. Pathfinder experienced no more system resets or loss of data.

PROBLEMS

10.1 Develop a functional, physical, and allocated architecture for an OSI-compliant communication system using the material presented in this chapter for the OSI reference model. Note the physical architecture of the communication system will include the physical communication network as well as the layers of the OSI reference model.

10.2 Develop a functional, physical, and operational model for a CORBA-compliant software system. Use a physical architecture comprised of the IDL, ORB, CORBAservices, CORBAfacilities, and CORBAdomains.

10.3 Select several items for your OnStar project from previous chapters and design an interface for those items.

10.4 Select several items for your ATM project from previous chapters and design an interface for those items.