Chapter 12: Distributed Object Technology

Distributed object technology provides the mechanism for linking objects in a network. With distributed objects, users can concentrate on what they need instead of how to get it. This chapter covers environments, common object request broker architecture (CORBA), the common object model, and the distributed system object model.

Distributed Object Environment

The distributed object environment (DOE) is a mechanism for objects to access one anothers public state and export functionality. CORBA is technically superior to the DOE, but many vendors claim that the DOE is superior when based on Java.

The DOE is useful in client/server development and Internet applications. Developers put together complex client/server systems simply by assembling and extending reusable distributed objects. Distributed objects can be altered without changing their core functionality. Distributed objects allow the developer to specify self−defined and strongly typed entities. Thus developers can mix and match reusable assets even if the assets reside across the network.

Intranets, Extranets, and the Internet have contributed to the growth of object technology by requiring the need to better manage object distribution throughout the enterprise. An Intranet is an electronic business (e−business) application within an organization. An Extranet is an e−business application deployed within the larger community of an organization and includes its suppliers, vendors, and contractors. Connecting to the Internet makes the information it contains accessible from anywhere in the world using ubiquitous browser software. Intranets, Extranets, and the Internet require the management of distributed transactions regardless of platform, language, and scalability. The World Wide Web (WWW) is a highly advanced distributed object system. Figure 12−1 shows a graphic view of the DOE.

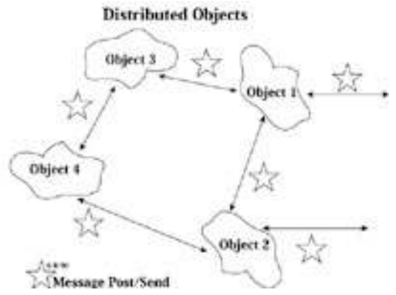

Figure 12−1: A sample of distributed object environment
The integration of the Web with CORBA or the common object model (COM) provides an excellent opportunity for distributed object technology. CORBA and COM are server−centric architectures that place the state and behavior of objects in server processes called *shared libraries* or *repositories*. Clients can send requests to objects that logically reside elsewhere. Thus the DOE is an environment that allows objects to be distributed across a heterogeneous network and allows each computing component to interoperate as a whole. When objects are distributed across a network, clients can be servers, and conversely, servers can be clients.

Distributed Computing Environment

The distributed computing environment (DCE) combines object−oriented concepts with client/server technology to produce a framework for building modular and scalable distributed applications at a relative high level of abstraction. Distributed object technology provides the means to do so by enabling access to applications and data wherever servers are located. The DCE provides a standardized environment for distributed applications across platforms. The DCE is only a foundation.

In the DCE, hardware, software, and networking work together in a system. The developer prepares a graphic model of a system, reuses well−tested components, and identifies interfaces. The integration of a system saves time and money. The DCE is more cost−effective in multiple applications than in a single application.

DCE products are compatible with one another because the Open Software Foundation standardizes and licenses most of the core services. Developers can standardize DCE products on disparate platforms, and they will still work together.

Common Object Request Broker Architecture

CORBA is an object−oriented standard produced by the Object Management Group (OMG). The standard has defined many key areas of distributed computing. The CORBA standard includes mapping between an interface definition language (IDL) and C++, Java, Smalltalk, and Ada95. The OMG works with Microsoft so that CORBA and COM provide standard architecture for distributed object computing. Figure 12−2 shows a graphic view of the structure of CORBA.

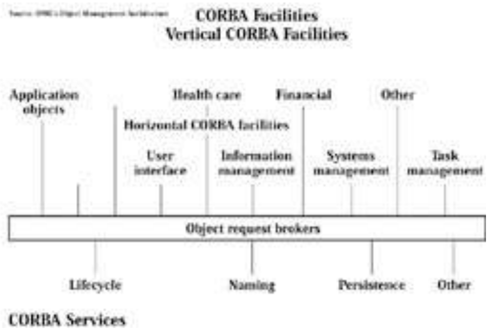

Figure 12−2: An overview of CORBA

CORBA solves interoperability problems. Each node is an object with a well−defined interface that is identified by a unique handling technique. The messages are passed between a sending object and a target object. The target object is also identified by its unique handling technique. The message format is defined in an interface known to the system. CORBA connects objects only and not applications. The OMG provides enterprise integration in object management architecture (OMA), which describes the overall architecture for distributed object computing based on an object request broker (ORB). CORBA specifies a messaging facility for DOE. CORBA consists of the following four major parts:

1. **Application objects.** The developer creates application objects specific to the applications. These exist in support of a particular application. The developer defines these objects using IDL so that the objects can communicate with other CORBA−compliant ORBs.
2. **ORB.** The ORB is the center of all activities. The ORB is middleware that establishes client/server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass in the parameters, invoke its method, and return the result. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an objects interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

The ORB is a piece of infrastructure technology that helps in the development of CORBA objects and arranges for objects to access one another at run time as defined by the CORBA specification. The ORB is an engine that can communicate with other local or remote objects by using a common interface and network protocol. An ORB can make requests to other ORBs, and they can also process responses. All of these processes happen behind the scenes, hidden from the user and the client/server applications.

The ORB ensures portability and interoperability of all application objects that exist on various connected computers. In fielding typical client and server applications, developers use their own design or a recognized standard to define the protocol to be used between the devices. The protocol definition depends on the implementation language, network transport, and several other factors. The ORB simplifies this process. With an ORB, the protocol is defined through the application interfaces via a single implementation language−independent specification, the IDL.

The ORB also provides flexibility by letting programmers choose the most appropriate operating system, execution environment, and programming language for each component of a system under construction. The ORB allows the integration of existing components. In an ORB−based solution, developers model the legacy component using the same IDL that they use for creating new objects. They then write a wrapper code that translates between the standardized bus and the legacy interfaces.

3. **CORBA facilities** provide services to applications and consist of two subpartsvertical and horizontal. These are a collection of services that relate more to clients than to the server. The vertical CORBA facilities define higher−level objects and interfaces that include domain−specific vertical services. The horizontal CORBA facilities are domain−independent horizontal services.
4. **CORBA services** provide a group of services that use object interfaces to define interfaces for general purpose supporting services. They provide a base set of services encapsulated within the ORB. The object services augment the base functionality of the ORB.

CORBA Benefits

- CORBA allows applications to communicate with one another no matter where they are located or who has designed them.
- The entire system is self−describing.
- The specification of a service is always separated from the implementation.
- CORBA 2.0 provides enough detail, especially considering the object−to−object communications links, to enable two ORBS from two or more vendors to work together.
- The user discovers objects and reuses them.
- The CORBA concept relates to domain and subdomain relationships.
- The CORBA specification includes how objects are defined, created, dispatched, and invoked and how they communicate with one another.

Interface Definition Language

The IDL defines interfaces to objects and defines the interface of each object in a unified way. The ORB interfaces are defined via an IDL. The interface definition specifies the operations that the objects are prepared to perform, the input and output parameters that they require, and any exceptions that may be generated along the way. Remote objects view a CORBA object purely in terms of this interface. The IDL is accessible to objects written in a cross−platform communications architecture in any language. The CORBA architecture separates the interface, written in OMG IDL, from the implementation, which must be written in a programming language.

To the client or user, the interface represents a promise that when the client sends a proper message to the interface, the response will come back. To the object implement, the interface represents an obligation that he or she must implement, in some programming language, all of the operations specified in the interface.

CORBA Architecture

In addition to the ORB and IDL, the major components of CORBA are as follows:

- The interface repository contains all IDL definitions that describe the attributes, operations, user−defined types, and exceptions of the server objects.
- The basic object adapter (BOA) represents the interface between the ORB and the server applications. It dispatches objects that the server applications maintain and exchanges messages with the server.
- The static invocation interface (SII) is a stub−based interface used by client programs to invoke service on application objects.
- The dynamic invocation interface (DII) is a generic interface that does not require stubs but rather supports dynamic construction of object invocations by the client program at run time.
- The object reference is the value that unambiguously identifies an object. Object references are never reused to identify another object.

Common Object Model

The COM is a part of Microsofts Windows family of operating systems. The COM defines a set of base classes and a mechanism for constructing and dynamically linking objects. The COM provides the underlying object model for components. All COM objects are required to implement behavior that allows for the software component plug and play. Microsoft engineered the COM with distribution in mind, and it initially supported communications on a single machine.

The COM provides a framework for development and deployment of software components. The developers capture abstractions as component interfaces and then provide binary classes that implement those interfaces. The COM includes an IDL derived from the open group DCE IDL.

The COM and CORBA both provide encapsulation by allowing client access to an object only through the objects interface. The COM and CORBA provide polymorphism, the ability for different kinds of objects to respond approximately to the same message. The COM and CORBA also support inheritance theoretically, but in reality the COM does not directly implement inheritance. The COM is a part of the framework of object linking and embedding (OLE) and ActiveX.

Distributed COM

The distributed common object model (DCOM) is a Microsoft object broker and is part of Microsoft Windows NT 4.0. The DCOM is an extension of the COM, extending the programming model introduced by the COM to work across the network. The DCOM provides communication across networks using remote procedures call (RPC). The future version of the DCOM is going to support other protocols. These extensions include improved location and packaging transparency, additional threading modeling, new security options, and additional administration capabilities.

Clients can access DCOM objects on remote machines without specific coding for the network. Alternatively, clients can specify the remote host that should be accessed when creating a new DCOM object.

The DCOM uses the same mechanism as CORBA IDL but with a different nameobject definition language (ODL). The DCOM supports distributed application development and provides directory, security, transaction, and processing support. With its mechanisms for placement of parameters, client proxies, and server stubs, the DCOM distributes COM object infrastructure to support client requests made to objects on remote computers as shown in Figure 12−3.

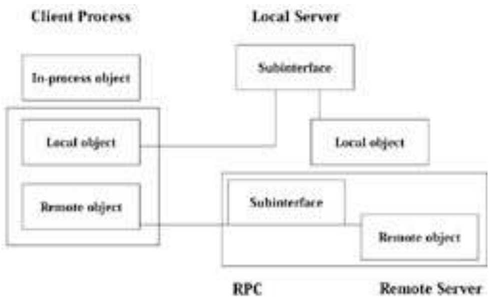

Figure 12−3: DCOM client server application overview

The DCOM is an OLE−enabled infrastructure for distributed development. The DCOM provides developers and users with access to data from other OLE−enabled applications. Linked applications can share data, whereas embedded applications actually commit their data to applications that access them. The basic OLE concept formed the foundation of the COM, which specifies how to build COM and OLE components and distributed objects, including the application program interfaces (APIs) and protocols that link COM/OLE components and distributed objects. The COM also specifies naming conventions, security, and directory services for object−to−object communications. The COM defines how components are built and interact; the DCOM defines distribution and communications with COM−enabled objects on the network.

The DCOM itself is not a development environment but rather a means of distributing application services. Developers can build servers for distribution using any COM−enabled development tool for ActiveX. Once the ActiveX exists, the DCOM takes over. A DCOM server can reside anywhere. Developers need only reference the server in the client applications, and the DCOM can find the correct resource for the client.

ActiveX

ActiveX is compatible with Microsoft Internet Explorer 3.0 or later versions that Microsoft offers free of charge. Microsoft designed ActiveX as a simplification of the COM to support linking of objects and applications across the network. ActiveX is widely discussed as a means of dynamically loading applets across the network, but it actually supports client or server side objects. ActiveX can be used interchangeably with browsers or standalone applications.

The most widely used ActiveX objects are so−called controls that implement user interface components; however, an ActiveX object need not have a user interface. An ActiveX control is a COM server in process. ActiveX components exist as precompiled binary objects suitable only for a specific platform.

CORBA 2.0 and Interoperability

All CORBA 2.0compliant products provide interoperability among one another using Internet inter−ORB protocol (IIOP). IIOP provides standardized protocol for all CORBA−compliant ORBs. This enables object development using ORBs from different vendors to interoperate transparently. The IIOP communication reduces network traffic while expanding the services that can be offered by a Web browser. The OMG has adopted a formal specification for COM/CORBA *interworking*. An OMG COM/CORBA committee created the term interworking to distinguish interoperability between technologies from inter−ORB interoperability.

Building distributed applications requires a proven infrastructure that scales to handle the diversity of the enterprise or the Internet. CORBA provides that infrastructure, allowing the developer to build standards−based systems from components implemented in a variety of languages and running on many different platforms. CORBA guarantees interoperability between these components through use of a standardized wire−protocol (IIOP). The IIOP defines a common data representation and a set of request and reply messages. The ORB from a single vendor has no problem communicating from clients to servers across a network. The ORB from different vendors can also interoperate with IIOP.

On the Active platform, the DCOM is the plumbing and wiring that connect COM objects into distributed applications. The DCOM acts as part of the operating systems infrastructure, allowing ActiveX and Active Server objects to find each other over a network.

Distributed System Object Model

The distributed system object model (DSOM) is an IBM product and is an extension of the system object model (SOM). The SOM is an object−oriented technology for building, packaging, and manipulating binary class libraries. The SOM is language neutral. The SOM preserves the object−oriented characteristics of encapsulation, inheritance, and polymorphism. Major benefits of SOM are that the changes to SOM class do not require source code changes in client programs and those client programs need not be recompiled. The SOM conforms the OMG CORBA standard. With the SOM, class implementers describe the interface for a class of object in CORBA IDL.

CORBA 3.0

CORBA 3.0 will include IDL mappings for developers writing in COBOL, Java, or a fourth−generation language, such as Sybase division Powersofts PowerBuilder, to access and generate CORBA objects without wrestling with the complex IDL. A CORBA component model would allow developers to create components visually on a project palette and access methods and properties more easily than the coding−intensive approach required by IDL. The component model will be based largely on JavaBeans. JavaScript will be among the scripting languages considered according to the chairman of the OMG.

The OMG plans to bolster its forthcoming CORBA 3.0 specification with a DOM for building components that is designed to head off DCOM. The CORBA 3.0 addition, tentatively called *CORBABeans,* aims to enable the creation of lightweight, reusable, graphic−based components that can be distributed across an enterprise. The CORBA 3.0 specification will provide mapping from Java to C++ objects and hide CORBA IDL to ease programming. The OMG has proposed three stipulations to the CORBABeans specification:

1. The specification should equally address the needs of the Java community and the C++ community.
2. The Java−related features should be pertinent and not brought into CORBA for no apparent reason.
3. CORBA 3.0 must be compatible with earlier versions of CORBA.

Case Study 12−1

The Advanced Trading Corporation (ATC) of New York is interested in buying a special type of sugar, Brown Sugar, from the international market. To gain the competitive edge, the company used distributed object technology as shown in Figure 12−4. The ATC has many branch offices throughout the world.

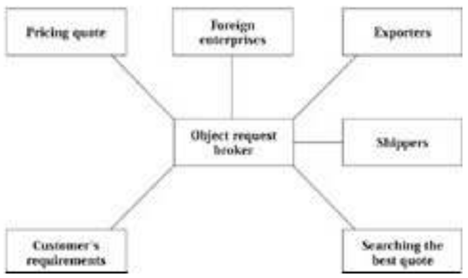

Figure 12−4: Logical flow with distributed object technology

The ATC is using C++ as the object−oriented language for their inventory system at their headquarters in New York. The system runs in a UNIX environment. The shipping system was developed by the ATC shipping agent in Singapore in the COBOL language and runs on a mainframe. The ATCs branch office in Los Angeles is monitoring current pricing on the Internet from all over the world and maintains this information in a Microsoft Excel spreadsheet. The new order processing has been developed in Smalltalk and is run on a powerful workstation by the purchasing agent at the ATC headquarters.

This application combines different hardware and software technologies. However, neither application developers nor users care about machines, locations, or programming languages. They view the world in terms of distributed objects.

Case Study 12−2: Management of a Bank Account

A user of an account object will wish to make deposits and withdrawals. An account object will also need to hold the balance of the account and perhaps the name of the accounts owner.

An example interface is as follows:

```
//IDL

interface account {

//Attributes to hold the balance and the name

//of the accounts owner.

Attribute float balance;

Readyonly attribute string owner;

//The operations defined on the interface.

Void makeDeposit (in float amount,

    • Out float newBalance);

      Void makeWithdrawal (in float amount,

          ♦ Out float newBalance);
      };
```

The account interface defines the attribute balance and owner, which are properties of an account object. The attribute balance may take values of type float, which is one of the basic types of IDLs and represents a floating point type. The attribute owner is of type string and is defined to be readyonly.

Two operations, makeDeposit () and makeWithdrawal (), are provided. Each has two parameters of type float. Each parameter must specify the direction in which the parameters are passed. The possible parameter modes are as follows:

- ***In.*** The parameter is passed from the caller (client) to the called object.
- ***Out.*** The parameter is passed from the called object to the caller.
- ***In−out.*** The parameter is passed in both directions.

Here, amount is passed as an in−parameter to both functions, and the new balance is returned as an out−parameter. The parameter−passing mode must be specified for each parameter, and it is used to improve the self−documentation of an interface and help guide the code to which the IDL is subsequently translated.

Line comments are introduced with the character // as shown. Comments spanning more than one line are delimited by /* and */.

Multiple IDL interfaces may be defined in a single source file, but it is common to define each interface in its own file (Orbix by IONA).

Case Study 12−3: A−Bank Information System

To remain competitive, A−Bank began offering products such as mutual funds and brokerage accounts. The customers expected to receive a single unified statement listing the balances and transactions for all of their accounts, including checking and savings accounts, mortgages, credit cards, brokerages, and retirement accounts. A−Bank realized that to meet customers demand, they needed to change from an account focus to a customer focus. This would result in customers expecting their banking activities to be structured in terms of their overall relationship with the bank rather than in terms of a single account that they may own.

However, like most banks, A−Banks computer systems were optimized for account processing rather than for providing integrated customer data. Mainframe computers managed direct deposit accounts, and a Digital

VAX/VMS system tracked mutual funds. Meanwhile, A−Bank outsourced brokerage account processing to a vendor who used a Tandem system. As a result, bank employees were required to log into several different computer systems, all with character−cell user interfaces. What should have been routine transactions for customers required bank employees to navigate three or more disparate information systems as shown in Figure 12−5. A simple question from a customer, such as How much money do I have in all of my accounts combined? could require an employee to go to several different systems of record, none of which offered a graphic user interface. This impeded efficient customer service.

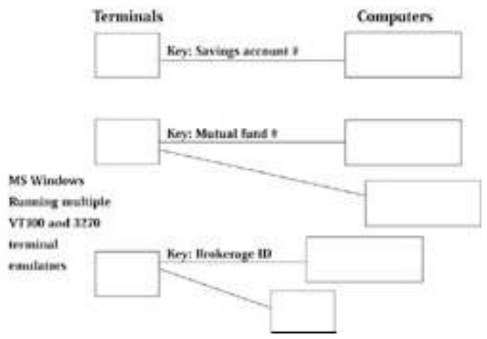

Figure 12−5. Before information access

A−Bank began to seek remedies to this problem. Reengineering of all of the banks systems of record to embrace client/server or any other technology would be neither practical nor cost−effective. Too much investment already existed in the legacy systems. A−Bank needed a way to overcome the inherent disparity between the existing systems. They needed to seamlessly integrate these systems functionality and preferably deliver the integrated result through a graphic user interface. A key objective was to deliver the integrated functionality through a native Microsoft Windows user interface.

A−Bank consulted Digital Equipment Corp. The Digital/Cushing Group Team recommended a three−tier client/server architecture to deliver a rapid solution that was production−grade quality. The Digital/Cushing Group Team also recommended the use of Digitals ORB technology to meet the goals. The ORB offers a way to construct distributed application systems using object−oriented semantics to define an application−level communication protocol between client and server programs. The selection of Digitals ORB directed the effort within A−Bank that would become the success story of the CORBA−based application development effort as shown in Figure 12−6. This technology enabled A−Bank to become the first bank to offer Internet−based access to account balances and several other innovations (Digital/Cushing Group Team; A−Bank is actually the Wells Fargo Bank).

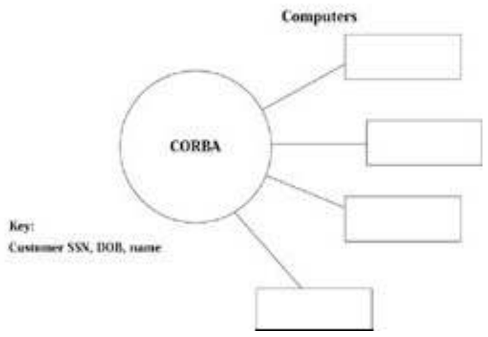

Figure 12−6: After information access