C H A P T E R 1 1

Overview of CORBA

In previous chapters we have seen how Java objects can be distributed and accessed using Java's RMI. Now we'll shift our attention to an alternative approach for distributing and accessing objects. This approach is based on OMG's CORBA, which is an industry standard for creating and using distributed objects.

This chapter presents an overview of CORBA, its architecture, and applications. We'll discuss the components of CORBA, its services and facilities, and the new features that are being added to CORBA that will be part of CORBA 3.0.

11.1 INTRODUCTION TO CORBA

CORBA, which stands for Common Object Request Broker Architecture, is an industry standard developed by the OMG (a consortium of more than 700 companies) to aid in distributed objects programming. CORBA is just a *specification* for creating and using distributed objects; CORBA is *not* a programming language.

The CORBA architecture is based on the object model. This model is derived from the abstract core object model defined by the OMG in the *Object Management Architecture Guide*, which can be found at http://www.omg.org. The model is abstract in the sense that it is not directly realized by any particular technology; this allows applications to be built in a standard manner using basic building blocks such as objects. Therefore, a CORBA-based system is a collection of objects that isolates the requestors of services (clients) from the providers of services (servers) by a well-defined encapsulating interface. It is important to note that CORBA objects differ from typical programming objects in three ways:

- CORBA objects can run on any platform.
- CORBA objects can be located anywhere on the network.
- CORBA objects can be written in any language that has IDL mapping.

11.2 CORBA ARCHITECTURE

The OMG's Object Management Architecture (OMA) tries to define the various high-level facilities that are necessary for distributed object-oriented computing. The core of the OMA is the Object Request Broker (ORB), a mechanism that provides object location tranparency, communication, and activation. Based on the OMA, the CORBA specification which provides a description of the interfaces and facilities that must be provided by compliant ORBs was released.

CORBA is composed of five major components: ORB, IDL, dynamic invocation interface (DII), interface repositories (IR), and object adapters (OA). These are discussed in the following sections.

11.2.1 The object request broker

The CORBA specification must have software to implement it. The software that implements the CORBA specification is called the ORB. The ORB, which is the heart of CORBA, is responsible for all the mechanisms required to perform these tasks:

- Find the object implementation for the request.

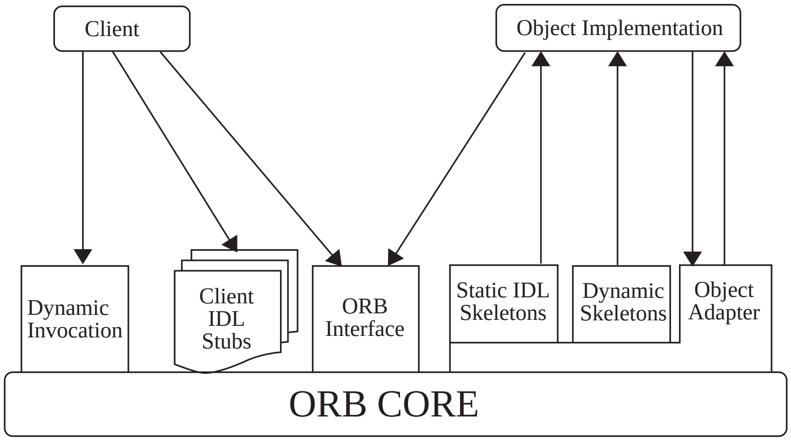

Figure 11.1 The structure of the CORBA 2.0 ORB

- Prepare the object implementation to receive the request.
- Communicate the data making up the request.

A number of implementations exist in the market today, including ORBIX from IONA Technologies (http://www.iona.ie), VisiBroker from Inprise (http://www.inprise.com), and JavaIDL from JavaSoft (http://java.sun.com/products/jdk.idl). Throughout this part of the book, we will use the VisiBroker ORB for Java, version 3.1. Figure 11.1 shows how the five major components of CORBA fit together.

Figure 11.2 shows a request being sent by a client to an object implementation. The client is the entity that wishes to perform an operation on the object, and the object implementation is the actual code and data that implements the object. Note that in this figure, the client, ORB, and object implementation are all on a single machine (meaning they're not separated by a network).

There are two important things to note about the CORBA architecture and its computing model:

- Both the client and the object implementation are isolated from the ORB by an IDL interface.

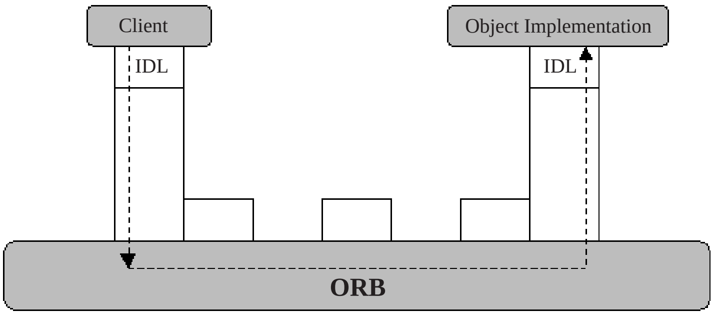

Figure 11.2 A request from a client to an object implementation

- All requests are managed by the ORB. This means that every invocation (whether it is local or remote) of a CORBA object is passed to an ORB. In the case of a remote invocation, however, the invocation passed from the ORB of the client to the ORB of the object implementation as shown in figure 11.3.

11.2.2 Different vendors and different ORBs

Since there is more than one CORBA implementation, and these implementations are from different vendors, a good question at this point would be whether objects implemented in different ORBs from different vendors would be able to communicate with each other. The answer is this: all CORBA 2.0 (and above) compliant ORBs are able to interoperate via the Internet Inter-ORB Protocol, or IIOP for short. This was not true for CORBA 1.0 products, however. The whole purpose of IIOP is to ensure that your client will be able to communicate with a server written for a different ORB from a different vendor.

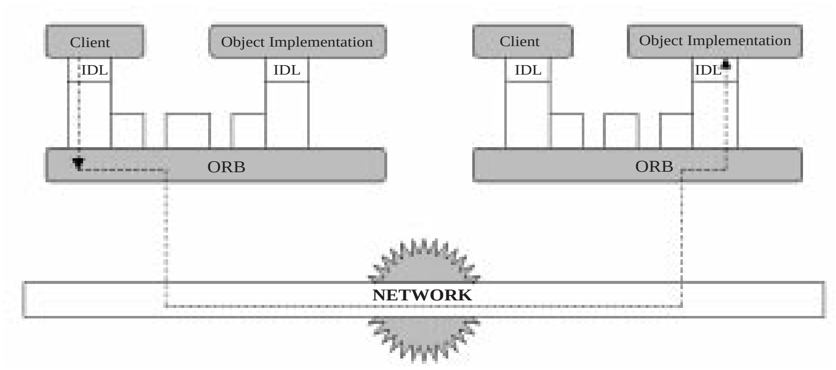

Figure 11.3 A request from a client to an object implementation within a network

11.2.3 Interface definition language

As with RMI, CORBA objects are to be specified with interfaces, which are the contract between the client and server. In CORBA's case, however, interfaces are specified in the special definition language IDL.

The IDL defines the types of objects by defining their interfaces. An interface consists of a set of named operations and the parameters to those operations. Note that IDL is used to describe interfaces only, not implementations. Despite the fact that IDL syntax is similar to C++ and Java, IDL is not a programming language.

Through IDL, a particular object implementation tells its potential clients what operations are available and how they should be invoked. From IDL definitions, the CORBA objects are mapped into different programming languages. Some of the programming languages with IDL mapping include C, C++, Java, Smalltalk, Lisp, and Python. Thus, once you define an interface to objects in IDL, you are free to implement the object using any suitable programming language that has IDL mapping. And, consequently, if you want to use that object, you can use any programming language to make remote requests to the object.

In chapter 13, I'll give you detailed coverage of IDL, and in chapter 14, I'll give you an overview of the IDL-to-Java mapping.

11.2.4 Dynamic invocation interface

Invoking operations can be done through either static or dynamic interfaces. Static invocation interfaces are determined at compile time, and they are presented to the client using stubs. The DII, on the other hand, allows client applications to use server objects without knowing the type of those objects at compile time. It allows a client to obtain an instance of a CORBA object and make invocations on that object by dynamically constructing requests. DII uses the interface repository to validate and retrieve the signature of the operation on which a request is made. CORBA supports both the dynamic and the static invocation interfaces.

11.2.5 Dynamic skeleton interface

Analogous to the DII is the server-side dynamic skeleton interface (DSI), which allows servers to be written without having skeletons, or compile-time knowledge, for the objects being implemented.

Unlike DII, which was part of the initial CORBA specification, DSI was introduced in CORBA 2.0. Its main purpose is to support the implementation of gateways between ORBs which utilize different communication protocols. However, DSI has many other applications beside interoperability. These applications include interactive software tools based on interpreters and distributed debuggers.

11.2.6 Interface Repository

The IR provides another way to specify the interfaces to objects. Interfaces can be added to the interface repository service. Using the IR, a client should be able to locate an object that is unknown at compile time, find information about its interface, then build a request to be forwarded through the ORB.

11.2.7 Object adapters

An object adapter is the primary way that an object implementation accesses services provided by the ORB. Such services include object reference generation and interpretation, method invocation, security of interactions, and object and implementation activation and deactivation.

11.3 CLIENT AND OBJECT IMPLEMENTATIONS

A distributed application consists of objects running within clients and servers. Servers provide objects to be used by clients as well as other servers. A client of an object has access to the object reference, which is the information needed to specify an object within an ORB; with that access, it can invoke operations on the object. A client knows the object through an interface, therefore the client knows nothing about how the object is implemented. A client generally sees objects and ORB interfaces through a language mapping.

11.4 OBJECT SERVICES

The OMA, which is shown in figure 11.4, is the next higher level that builds upon the CORBA architecture. The goal of the OMA is to allow applications to provide their basic functionality through a standard interface.

As you can see from figure 11.4, the OMA consists of two main components: *CORBAservices* and *CORBAfacilities*. CORBAservices provides basic services that almost every object needs; this includes naming service and event service. The CORBAfacilities provides higher-level functionality at the application level. As demonstrated in figure 11.4, CORBAfacilities are further divided into horizontal CORBAfacilities and vertical CORBAfacilities. Horizontal CORBAfacilities include user interface, information management, systems management, and task Management. Vertical (or Domain) CORBAfacilities are domain-based and provide functionality for a specific domain such as telecommunications, electronic commerce, or health care.

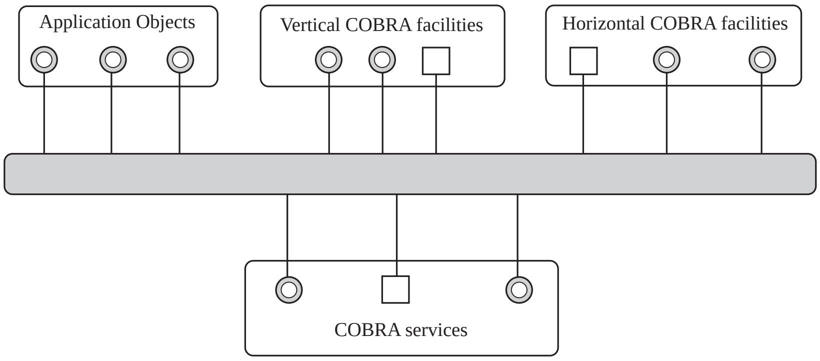

Figure 11.4 Object Management Architecture (OMA)

Object services are interfaces which are widely available and are most commonly used to support building well-formed applications in a distributed object environment built on a CORBA-compliant ORB. OMG object services have the following features:

- Objects may use few or many object services.
- The operations provided by object services are specified in IDL.

There have been a number of Request For Proposals (RFPs) issued for a set of specifications known as Common Object Services Specification, Volumes 1 and 2 (COSS 1 and COSS 2). They both include services for object naming, object events, object lifecycle, persistent object, object relationships, object externalization, object transactions, object concurrency, and object security. All of these services are now known as CORBA Services. Here is a description of some of these services.

- *Object naming service* This service supports a name-to-object association called a name binding, which is always defined relative to a naming context. Different names can be bound to an object in the same or different context at the same time. This service supports a number of operations, including `bind`, `unbind`, and `lookup`.
- *Event service* This service supports notification of events to interested objects. It provides asynchronous communications between cooperating, remote objects.
- *Persistent object service* This service provides common interfaces for the mechanisms used for retaining and managing the persistent state of objects in a data-store independent manner. Of course, the object has the responsibility of managing its state, but it can use or delegate to this service for the actual work.
- *Concurrency control service* This service defines how an object mediates simultaneous access by one or more clients, so that the objects it accesses remain consistent and coherent.

11.5 NEW FEATURES IN CORBA 3.0

Since its inception in 1991, CORBA has had to evolve to remain viable as a basis for distributed applications. As part of this continuing evolution, several significant new features are being added to CORBA that will be part of CORBA 3.0. The new features include Portable object adapter (POA), CORBA messaging, and objects by value. A brief overview of these new CORBA features is provided here.

11.5.1 Portable object adapter

Object adapters mediate between CORBA objects and programming language implementations (servants). Object adapters provide a number of services, including the creation of CORBA objects and their references, dispatching requests to the appropriate servant that provides an implementation for the target object, and activation and deactivation of CORBA objects.

In CORBA 2.0, the only standard object adapter defined by the OMG is called the basic object adapter (BOA), which only provides basic services to allow a variety of CORBA objects to be created. ORB vendors and developers, however, discovered that the BOA is ambiguous and missing some features. This led vendors to develop their own proprietary extensions, which resulted in poor portability between different ORB implementations.

The new standard object adapter, the POA, provides new features that allow applications and their servants to be portable between different ORBs supplied by different vendors. This new adapter provides a great deal of flexibility for server implementations. As I mentioned earlier, the POA mediates between the ORB and the server application. Figure 11.5 shows a request from a client to a server.

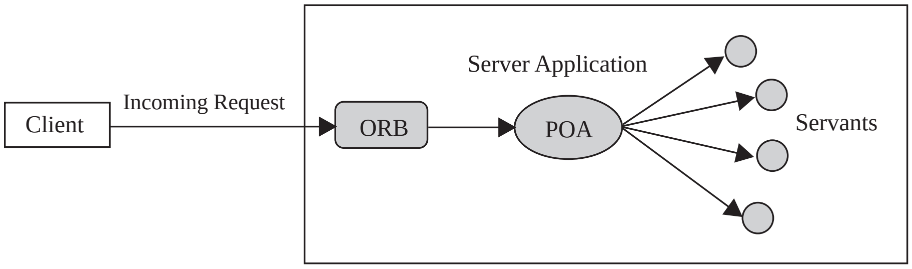

Figure 11.5 Request dispatching based on POA

The client invokes the request using a reference that refers to the target object. The request is then received by the ORB, which will dispatch the request to the POA that hosts the target object. The POA will then dispatch the request to the servant, which subsequently carries the request and sends the results back to the POA, to the ORB, and finally to the client. Note that an application may have multiple POAs, and in order for the ORB to dispatch the request to the right POA, it uses an *object key*, which is an identifier that is part of the request that is kept in the object reference. A part of the object key called the object ID is used by the POA to determine an association between the target object and a servant.

To summarize, the POA deals mainly with three entities: the *object reference*, the *object ID* (both of which are used for identification), and the *servant*, which actually implements CORBA objects.

11.5.2 CORBA messaging

CORBA 2.0 provides three different techniques for operation invocations:

- *Synchronous* The client invokes an operation, then pauses, waiting for a response.
- *Deferred synchronous* The client invokes an operation then continues processing. It can go back later to either poll or block waiting for a response.
- *One-way* The client invokes an operation, and the ORB provides a guarantee that the request will be delivered. In one-way operation invocations, there is no response.

Synchronous invocation techniques tend to tightly couple clients and servers. This has led many people to criticize CORBA as being unable to cope with large distributed systems. For this reason, a new specification (the CORBA messaging specification, which can be found at http://www.omg.org), has been adopted by the OMG. This new specification preserves the invocation techniques in CORBA 2.0 and adds two new asynchronous request techniques:

- *Callback* The client supplies an additional object reference with each request invocation. When the response arrives, the ORB uses that object reference to deliver the response back to the client.
- *Polling* The client invokes an operation that immediately returns a `valuetype` that can be used to either poll or wait for the response.

The callback and polling techniques are available for clients using statically typed stubs generated from IDL interfaces. These new techniques represent a significant advantage for most programming languages because static invocations provide a more natural programming model than the DII.

11.5.3 Objects by value

One of the ongoing criticisms of CORBA 2.0 is the lack of support for passing objects by value. This has been addressed by adding support for passing objects by value (see the document *Objects By Value*, which can be found at http://www.omg.org). This has led to the addition of a new construct to the OMG IDL called the `valuetype`. A `valuetype` supports both data members and operations, much the same as a Java class definition.

When a `valuetype` is passed as an argument to a remote operation, it will be created as a copy in the receiving address space. The identity of the `valuetype` copy is separate from the original, so operations on one have no effect on the other. It is important to note that operations invoked on `valuetypes` are local to the process in which the `valuetype` exists. This means that `valuetype` invocations never involve the transmission of requests and replies over the network.

11.6 SUMMARY

- CORBA stands for Common Object Request Broker Architecture. CORBA is an industry-standard developed by the OMG, a consortium of more than 700 companies.

CORBA is not a programming language; it's a specification for creating and using distributed objects.

- CORBA objects are different from typical programming objects in three ways: CORBA objects can run on any platform, they can be located anywhere on the network, and they can be written in any language that supports IDL mapping.
- CORBA is composed of five major components: ORB, IDL, DII, IR), andOA.
- The ORB is responsible for finding the object implementation for a request, preparing the object implementation to receive the request, and communicating the data making up the request.
- The OMA is the next higher level that builds upon the CORBA architecture. OMA consists of two main components: CORBAservices and CORBAfacilities. The OMA allows applications to provide their basic functionality through a standard interface.
- CORBA 3.0 will have several major new features, including POA, CORBA messaging, and objects by value. POA provides new features that allow applications and their servants to be portable between different ORBs supplied by different vendors. CORBA messaging adds two new asynchronous request techniques: *polling* and *callback*. These new techniques represent a significant advantage for most programming languages because static invocations provide a more natural programming model than the DII. Using objects by value, it is now possible to pass objects by value with CORBA.