# CORBA

# In large, distributed applications, your needs might not be satisfied by the preceding approaches. For example, you might want to interface with legacy data stores, or you might need services from a server object regardless of its physical location. These situations require some form of Remote Procedure Call (RPC), and possibly language independence. This is where CORBA can help.

# CORBA fundamentals

# The object interoperability specification developed by the OMG is commonly referred to as the Object Management Architecture (OMA). The OMA defines two components: the Core Object Model and the OMA Reference Architecture. The Core Object Model states the basic concepts of object, interface, operation, and so on. (CORBA is a refinement of the Core Object Model.) The OMA Reference Architecture defines an underlying infrastructure of services and mechanisms that allow objects to interoperate. The OMA Reference Architecture includes the Object Request Broker (ORB), Object Services (also known as CORBA services), and common facilities.

# CORBA Interface Definition Language (IDL)

# CORBA is designed for language transparency: a client object can call methods on a server object of different class, regardless of the language they are implemented with. Of course, the client object must know the names and signatures of methods that the server object exposes. This is where IDL comes in. The CORBA IDL is a language-neutral way to specify data types, attributes, operations, interfaces, and more. The IDL syntax is similar to the C++ or Java syntax. The following table shows the correspondence between some of the concepts common to three languages that can be specified through CORBA IDL:

# Some CORBA services

# This is a short description of what the JavaIDL-related code is doing (primarily ignoring the part of the CORBA code that is vendor dependent). The first line in main( ) starts up the ORB, and of course, this is because our server object will need to interact with it. Right after the ORB initialization, a server object is created. Actually, the right term would be a transient servant object: an object that receives requests from clients, and whose lifetime is the same as the process that creates it. Once the transient servant object is created, it is registered with the ORB, which means that the ORB knows of its existence and can now forward requests to it.

# Java Applets and CORBA

# Java applets can act as CORBA clients. This way, an applet can access remote information and services exposed as CORBA objects. But an applet can connect only with the server from which it was downloaded, so all the CORBA objects the applet interacts with must be on that server. This is the opposite of what CORBA tries to do: give you complete location transparency.

# CORBA vs. RMI

# You saw that one of the main CORBA features is RPC support, which allows your local objects to call methods in remote objects. Of course, there already is a native Java feature that does exactly the same thing: RMI (see Chapter 15). While RMI makes RPC possible between Java objects, CORBA makes RPC possible between objects implemented in any language. It’s a big difference.

# CORBA and RMI/IIOP

CORBA 980CORBA fundamentals 981Java Applets and CORBA 989CORBA vs. RMI 989Use code written in other languages, running on other architectures. This is accomplished using the Common Object Request Broker Architecture (CORBA), which is directly supported by Java.CORBA is not a language feature; it’s an integration technology. It’s a specification that vendors can follow to implement CORBA-compliant integration products. CORBA is part of the OMG’s effort to define a standard framework for distributed, language-independent object interoperability.CORBA supplies the ability to make remote procedure calls into Java objects and non-Java objects, and to interface with legacy systems in a location-transparent way. Java adds networking support and a nice object-oriented language for building graphical and non-graphical applications. The Java and OMG object model map nicely to each other; for example, both Java and CORBA implement the interface concept and a reference object model.The naming service is one of the fundamental CORBA services. A CORBA object is accessed through a reference, a piece of information that’s not meaningful for the human reader. But references can be assigned programmer-defined, string names. This operation is known as stringifying the reference, and one of the OMA components, the Naming Service, is devoted to performing string-to-object and object-to-string conversion and mapping. Since the Naming Service acts as a telephone directory that both servers and clients can consult and manipulate, it runs as a separate process. Creating an object-to-string mapping is called binding an object, and removing the mapping is called unbinding. Getting an object reference passing a string is called resolving the name.Again, the Naming Service specification is part of CORBA, but the application that implements it is provided by the ORB vendor. The way you get access to the Naming Service functionality can vary from vendor to vendor. The code shown here will not be elaborate because different ORBs have different ways to access CORBA services, so examples are vendor specific. (The example below uses JavaIDL, a free product from Sun that comes with a light-weight ORB, a naming service, and an IDL-to-Java compiler.) In addition, since Java is young and still evolving, not all CORBA features are present in the various Java/CORBA products.The first step is to write an IDL description of the services provided. This is usually done by the server programmer, who is then free to implement the server in any language in which a CORBA IDL compiler exists. The IDL file is distributed to the client side programmer and becomes the bridge between languages.//: c15:corba:ExactTime.idl//: c15:corba:RemoteTimeServer.javaimport org.omg.CORBA.\*; org.omg.CORBA.Object objRef = As you can see, implementing the server object is simple; it’s a regular Java class that inherits from the skeleton code generated by the IDL compiler. Things get a bit more complicated when it comes to interacting with the ORB and other CORBA services.To bind the servant object with a stringified object reference, we first create a NameComponent object, initialized with “ExactTime,” the name string we want to bind to the servant object. Then, using the rebind( ) method, the stringified reference is bound to the object reference. We use rebind( ) to assign a reference, even if it already exists, whereas bind( ) raises an exception if the reference already exists. A name is made up in CORBA by a sequence of NameContexts—that’s why we use an array to bind the name to the object reference.//: c15:corba:RemoteTimeClient.javaimport org.omg.CORBA.\*; org.omg.CORBA.Object objRef = This is a simple example, designed to work without a network, but an ORB is usually configured for location transparency. When the server and the client are on different machines, the ORB can resolve remote stringified references using a component known as the Implementation Repository. Although the Implementation Repository is part of CORBA, there is almost no specification, so it differs from vendor to vendor.As you can see, there is much more to CORBA than what has been covered here, but you should get the basic idea. If you want more information about CORBA, the place to start is the OMG Web site, at www.omg.org. There you’ll find documentation, white papers, proceedings, and references to other CORBA sources and products.This approach requires you to write a kind of integration layer, which is exactly what CORBA does for you, but then you don’t need a third-party ORB.Suppose you need to develop a multi-tiered application to view and update records in a database through a Web interface. You can write a database application using JDBC, a Web interface using JSP/servlets, and a distributed system using CORBA/RMI. But what extra considerations must you make when developing a distributed object system rather than just knowing API’s? Here are the issues:Java Naming and Directory Interface (JNDI) is used in Enterprise JavaBeans as the naming service for EJB Components on the network and other container services such as transactions. JNDI maps very closely to other naming and directory standards such as CORBA CosNaming and can actually be implemeted as a wrapper on top of it. JTA/JTS is used in Enterprise JavaBeans as the transactional API. An Enterprise Bean Provider can use the JTS to create transaction code, although the EJB Container commonly implements transactions in EJB on the EJB components’ behalf. The deployer can define the transactional attributes of an EJB component at deployment time. The EJB Container is responsible for handling the transaction whether it is local or distributed. The JTS specification is the Java mapping to the CORBA OTS (Object Transaction Service)Use of CORBA and RMI/IIOP in Enterprise JavaBeans is implemented in the EJB Container and is the responsibility of the EJB Container provider. Use of CORBA and RMI/IIOP in the EJB Container is hidden from the EJB Component itself. This means that the Enterprise Bean Provider can write their EJB Component and deploy it into any EJB Container without any regard of which communication protocol is being used.The Remote interface is a Java Interface that reflects the methods of your Enterprise Bean that you wish to expose to the outside world. The Remote interface plays a similar role to a CORBA IDL interface. BMP also gives flexibility where a CMP implementation may not be available. For example, if you wanted to create an EJB that wrapped some code on an existing mainframe system, you could write your persistence using CORBA.Because an EJB component is a distributed object, the deployment process should also create some client stubs for calling the EJB component. These classes should be placed on the classpath of the client application. Because EJB components can be implemented on top of RMI-IIOP (CORBA) or RMI-JRMP, the stubs generated could vary between EJB Containers; nevertheless they are generated classes.In this example the client program is a simple Java program, but you should remember that it could just as easily be a servlet, a JSP or even a CORBA or RMI distributed object.The sequence of the example is explained in the comments. Note the use of the narrow( ) method to perform a kind of casting of the object before a Java cast is performed. This is very similar to what happens in CORBA. Also note that the Home object becomes a factory for PerfectTime objects.One important consequence of Jini’s architecture is that the network protocol used to communicate between a proxy service object and a remote server does not need to be known to the client. As illustrated in the figure below, the network protocol is part of the service’s implementation. This protocol is a private matter decided upon by the developer of the service. The client can communicate with the service via this private protocol because the service injects some of its own code (the service object) into the client’s address space. The injected service object could communicate with the service via RMI, CORBA, DCOM, some home-brewed protocol built on top of sockets and streams, or anything else. The client simply doesn’t need to care about network protocols, because it can talk to the well-known interface that the service object implements. The service object takes care of any necessary communication on the network. Java Programming with CORBA, by Andreas Vogel & Keith Duddy (John Wiley & Sons, 1997). A serious treatment of the subject with code examples for three Java ORBs (Visibroker, Orbix, Joe). CORBA · 980RMI: AlreadyBoundException · 978; and CORBA · 989; bind( ) · 976; localhost · 977; rebind( ) · 978; Remote · 974; remote interface · 974; Remote Method Invocation · 973; remote object registry · 976; RemoteException · 974, 980; rmic · 979; rmic and classpath · 979; rmiregistry · 976; RMISecurityManager · 976; Serializable arguments · 978; skeleton · 978; stub · 978; TCP/IP · 977; unbind( ) · 978; UnicastRemoteObject · 974