

Comparing RMI, DCOM, & CORBA

Paul Augustin is NOLA's vice president of operations. He is currently assigned as the contractor assistant general manager for systems architecture and engineering at the SPAWAR Information Technology Center in New Orleans. In this role, he is responsible for the direction of requirements analysis, architecture, design, and testing for Navy personnel systems as well as technical support that the SPAWAR ITC is providing to other federal agencies.

In our last issue, we looked at Distributed Object Computing to integrate heterogeneous applications. If you read "Flexibility Using Distributed Object Computing" (Peer to Peer, spring 1999), you'll recall that DOC extends an object-oriented system by providing a means to distribute objects across a network, allowing each component to interoperate as a unified whole. Objects look "local" to applications, even though they are distributed to different computers throughout a network.

In "Flexibility," we focused on the Common Object Request Broker Architecture model, but other approaches are available as well, including the Distributed Component Object Model and Java Remote Method Invocation. Here we'll compare all three models, and explain how each model invokes a remote method, also known as **remoting**.

First, a note on remoting. To invoke a remote method, the client makes a call to the client-side proxy. The client-side proxy packs the call parameters into a request message and invokes a wire protocol to ship the message to the server. At the server side, the wire protocol delivers the message to the server-side stub. The

server-side stub then unpacks the message and calls the actual method on the object. In both CORBA and Java RMI, the server stub is called the skeleton and client stub is called the stub or proxy. In DCOM, the client stub is called the proxy and the server stub is called the stub.

CORBA

CORBA depends on an Object Request Broker^{3/4}a central bus over which CORBA objects interact transparently. CORBA uses Internet Inter-ORB Protocol for remoting objects. To request a service, a CORBA client acquires an object reference to a CORBA server object. The client then makes method calls on the object reference as if the CORBA server object resided in the client's address space. The ORB finds a CORBA object's implementation, prepares it to receive requests, communicates the requests, and carries the replies back to the client. CORBA can be used on a range of operating system platforms, from hand-held devices to mainframes.

DCOM

Think of DCOM as an extension of the Component Object Model, a

Microsoft framework that supports program component objects. DCOM supports remoting objects through a protocol named Object Remote Procedure Call. ORPC is a layer that interacts with COM's run-time services. A DCOM server is a body of code capable of serving up particular objects at run-time. Each DCOM server object supports multiple interfaces, each of which represent a different behavior of the object. A DCOM client calls into the exposed methods of a DCOM server by acquiring a pointer-to-server-object interface. The client object calls into the server object's exposed methods through the interface pointer, as if the server object resided in the client's address space.

Java RMI

RMI is the Java version of what is generally known as a remote procedure call, with the added ability to pass objects along with the request. The objects can include information that change the service performed in the remote computer. This property of RMI is often called "moving behavior." Java RMI relies on the Java Remote Method Protocol and on Java Object Serialization, which allows

objects to be transmitted as a stream. Since Java Object Serialization is specific to Java, both the Java RMI server object and the client object have to be written in Java.

Java RMI allows client/server applications to invoke methods across a distributed network of servers running the Java Virtual Machine. Although RMI is considered by many to be weaker than CORBA and DCOM, it offers such unique features as distributed automatic object management and has the ability to pass objects between machines.

The naming mechanism, RMIRegistry, runs on the server machine and holds information about available server objects. A Java RMI client acquires a reference to a Java RMI server object by looking up a server object reference and invoking methods on the server object, as if the Java RMI server object resided on the client. These server objects are named using Universal Resource Locators. A client acquires a reference by specifying the server object's URL, just as you would specify the URL to an HTML page.

While CORBA, DCOM, and Java RMI all provide similar mechanisms for

transparently accessing remote distributed objects, DCOM is a proprietary solution that works best in Microsoft environments. For an organization that has adopted a Microsoft-centered strategy, DCOM is an excellent choice. However, if any other operating systems are required in the application architecture, DCOM is probably not the correct solution. This may change as Microsoft attempts to make DCOM cross-platform compatible.

Because of its easy-to-use native-JAVA model, RMI is the simplest and fastest way to implement distributed object architecture. It's a good choice for RAD prototypes and small applications implemented completely in Java. Since RMI's native-transport protocol, JRMP, can only communicate with other Java RMI objects, it's not a good choice for heterogeneous applications.

CORBA and DCOM are similar in capability, but DCOM doesn't yet support operating system interoperability, which may discount it as a single solution. At the moment, CORBA is the logical choice for building enterprise-wide, open-architecture, distributed-object applications.