A Preview of CORBA 3

Jon Siegel, Object Management Group

The Common Object Request Broker Architecture (CORBA) from the Object Management Group has been the most influential standard in the OO world and one of the principal steps in the industry’s move to component-based development. Jon Siegel, Director of Domain Technology at the OMG and best-selling CORBA book author, gives us a preview of CORBA 3, the next step in the technology.

—Bertrand Meyer

CORBA 3, the first major addition to CORBA from the OMG since the Internet Inter-ORB Protocol (IIOP) added interoperability in 1996, will be released very soon.

In all, OMG specifications work at three levels: The Unified Modeling Language (UML) standardizes model representation in OO analysis and design; CORBA defines an interoperability infrastructure; and the Object Management Architecture (OMA) standardizes interfaces for services and facilities used in distributed applications from near system level (Naming and Event Services, for example) to application level (such as the Currency Object in finance, and the Patient or Person Identifier facility in health care).

I only have room to discuss CORBA in this column. For information on the rest, or to download any of the free OMG specifications, visit http://www.omg.org.

OMG INTERFACE DEFINITION LANGUAGE

In CORBA, programmers define their objects’ interfaces in OMG Interface Definition Language (IDL), which is also an international standard (ISO/IEC 14750 and ITU-T Recommendation X.920). When you write an interface definition, you specify the operations the object is prepared to perform, the input and output parameters each requires, and any exceptions that may be generated along the way. This interface constitutes a contract with clients of the object, who use the same interface definition to build and dispatch invocations the object implementation uses to receive them and respond. This design provides great flexibility and has many benefits. It enforces encapsulation and allows clients to access object implementations independent of each other’s programming language, OS, hardware platform, data representation, location on the network, and native protocol.

You can write your CORBA clients and objects in most standard languages. For six programming languages (soon to be seven), an OMG standard language mapping specifies how OMG IDL types and method invocations convert into language types and function calls or invocations. IDL compilers use the mappings to generate function or method calls. Currently, CORBA specifies OMG IDL language mappings for C, C++, Java, Cobol, Smalltalk, and Ada; a mapping for Lisp is being adopted. Not-yet-standard mappings are available now for Objective C, Visual Basic, Perl, and other languages.

CORBA OBJECT INVOCATIONS

Figure 1 shows a request passing from a client to an object implementation in the CORBA architecture. Two aspects of this architecture stand out.

First, both client and object implementation are isolated from the Object Request Broker (ORB) by their IDL interface, which is compiled into a stub (on the client end) and skeleton (on the object implementation end). Since clients see only the object’s interface, and never any implementation detail, the architecture guarantees substitutability of the implementation behind the interface—our plug-and-play distributed object software environment.

Second, the request does not pass directly from client to object implementation; instead, requests are always managed by an ORB. Every invocation of a CORBA object is passed to the ORB; the form of the invocation in the client is the same whether the target object is local or remote (if remote, the invocation passes from the ORB of the client to the ORB of the object implementation). Distribution details reside only in the ORB, where they are handled by software you bought, not software you built. Application code, freed of this administrative burden, concentrates on the problem at hand.
A SIMPLE ARCHITECTURE

A web of ORB-to-ORB communications pathways provides interoperability among all of the CORBA objects on a network. ORBs use the OMG-specified IIOP protocol for communication, and they use the Interoperable Object Reference (IOR) to pass object instance location information among themselves. From the client's point of view, the CORBA architecture is extremely simple: Every CORBA object is always running, from the time it is created until the time it is destroyed. Objects are identified by IORs; clients can make invocations on any valid object reference and expect the answer to come back. There are no client-accessible object activation or deactivation operations in CORBA.

It's impossible to convey the sophistication of the CORBA server-side architecture in the space of this column and have space left to describe anything else. OMG has defined a very powerful—yet standardized—object instance management infrastructure on the server side (called the Portable Object Adapter) that supports enterprise-scale numbers of objects and Internet-scale hit rates while maintaining the client's simple view of the encapsulated object implementation. The CORBA Component Model, a key part of CORBA 3, builds on the POA.

The specifications included in CORBA 3 divide neatly into three major categories: Java and Internet Integration; Quality of Service Control; and the CORBA component architecture.

JAVA AND INTERNET INTEGRATION

Three specifications enhance CORBA integration with Java and the Internet: Java-to-IDL mapping, the firewaller specification, and the interoperable naming service.

Java-to-IDL Mapping

CORBA 3 adds a Java-to-IDL mapping to the "normal" IDL-to-Java mapping that I described at the beginning of this column. The mapping defines an IDL interface for a Java object, with two effects: It lets Java programmers use the OMG standard protocol IIOP in remote invocations, and it allows Java servers to be invoked by CORBA clients written in any CORBA-supported programming language.

Firewall Specification

The CORBA 3 Firewall Specification defines transport-level firewalls, application-level firewalls, and (perhaps most interesting) a bidirectional General Inter-Orb Protocol (GIOP) connection useful for callbacks and event notifications. Transport-level firewalls work at the TCP level. By defining well-known ports 683 for IIOP and 684 for IIOP over SSL, the specification allows administrators to configure firewalls to cope with CORBA traffic over the IIOP protocol. There is also a specification for CORBA over Socks.

In CORBA, objects frequently need to call back or notify the client that invoked them; for this, the objects act as clients and the client-side module instantiates an object that is called back in a reverse-direction invocation. Because standard CORBA connections carry invocations only one way, a callback typically requires establishing a second TCP connection for this traffic heading in the other direction, which is a no-no to virtually every firewall in existence. Under the new specification, an IOP connection is allowed to carry invocations in the reverse direction under certain restrictive conditions that don't hurt the security of either site.

Interoperable Naming Service

The CORBA object reference is a cornerstone of the architecture. Before CORBA 3, the IOR was the only way to reach an instance and invoke it. There was no way to reach a remote instance—even if you knew its location and that it was up and running—unless you could get access to its object reference. The easiest way to do that was to get a reference to its Naming Service. But what if you didn't have a reference even for that?

The Interoperable Naming Service defines one URL-format object reference, iioptoc, that can be typed into a program to reach defined services at a remote location, including the Naming Service. A second URL format, iiorname, actually invokes the remote Naming Service using the name the user appends to the URL, and retrieves the IOR of the named object. For example, an iirname identifier iiorname://www.omg.org/NamingService would resolve to the CORBA Naming Service running on the machine whose IP address corresponds to the domain name www.omg.org (if we had one).

QUALITY OF SERVICE CONTROL

The new Asynchronous and Messaging Invocation (AMI) specification defines a number of asynchronous and time-independent invocation modes for CORBA and allows both static and dynamic invocations to use every mode. Asynchronous invocations' results can be retrieved by either polling or callback, with the choice made by the form used by the client in the original invocation.

Policies allow control of invocations' QoS. Clients and objects can control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live; set a start time and end time for time-
Component and Object Technology

sensitive invocations; and control routing policy and network routing hop count. Some of these controls only affect messaging-enabled CORBA installations, an optional part of the specification.

Minimum CORBA is primarily intended for embedded systems. Once they are finalized and burned into chips for production, embedded systems are fixed, and their interactions with the outside network are predictable. They have no need for the dynamic aspects of CORBA, such as the Dynamic Invocation Interface or the Interface Repository that supports it, which are therefore not included in minimum CORBA.

Real-time CORBA standardizes resource control—threads, protocols, connections, and so on—using priority models to achieve predictable behavior for both hard and statistical real-time environments. Dynamic scheduling, not a part of the current specification, is being added. Fault tolerance for CORBA is being addressed by an RFP, also in process, for a standard based on entity redundancy and fault management control.

CORBA COMPONENTS PACKAGE

Called valuetypes, objects passable by value add a new dimension to the CORBA architecture, which previously supported only passing (and invocation) by reference. Like conventional CORBA objects, valuetypes have state and methods; unlike CORBA objects, they do not (typically) have object references and are invoked in-process as programming language objects.

It is only when they are included in parameter lists of CORBA invocations that they show their talent of packaging up their state in the sending context, sending it over the wire to the receiving context, creating a running instance of the object there, and populating it with the transmitted state. Frequently used to represent nodes in binary trees or cyclically linked lists, valuetypes are specified and implemented to represent these important constructs faithfully.

One of the most exciting developments to come out of OMG since the IIOP protocol defined CORBA 2. CORBA components represent a multipronged advance with benefits for programmers, users, and consumers of component software. The three major parts of CORBA components are:

- a container environment that packages transactionality, security, and persistence and provides interface and event resolution;
- a component system that can be integrated with Enterprise JavaBeans; and
- a software distribution format that enables a CORBA component software marketplace.

For the programmer, the prepackaged CORBA components provides a higher level of abstraction than CORBA services, leveraging the expertise of business programmers who are not necessarily skilled at building transactional or secure applications. These programmers can now use their talents to produce business applications that acquire these attributes automatically.

After 10 years of cooperative work by OMG members, the base CORBA infrastructure is complete and in constant use at thousands of sites. The extensions bundled under the banner CORBA 3 bring ease-of-use and precise control to this established architecture.

All of us expect the result to be worthwhile. Although the IDL and CORBA services make CORBA accessible to programmers now, their low-level interfaces sometimes represent a barrier to business programmers who want to manipulate objects that look just like business entities. CORBA components and scripting will soon make it possible for these business users to assemble applications tailored precisely to their needs, while asynchronous invocation interfaces and QoS control allow sites to take advantage of networked facilities even where resources are stressed.

Jon Siegel is director of domain technology at the Object Management Group. Contact him at siegel@omg.org.

Related URLs

- CGI scripts and resources: http://desktoppublishing.com/cgi.html
- Oracle's Raw Iron initiative: http://oracle.com
- Servlet Central: http://www.servletcentral.com
- Active Server Pages: http://www.activesserverpages.com
- ASP resources: http://www.asp101.com
- Sun's JavaServer: http://jserv.javasoft.com/products/java-server
- IBM/Lotus Domino server: http://domino.lotus.com
- Microsoft's IIS: http://www.microsoft.com/ntservlet/Web
- Allaire Corporation: http://www.allaire.com
- Elemental Software: http://www.drumbeat.com

In the near future, even nonprogrammers will be able to build data-driven Web applications with new software development tools that are being introduced in the marketplace. One such tool is Elemental Software's Drumbeat 2000 development environment. Drumbeat 2000 uses ASPs to build Web applications. It applies an object-oriented approach to software development via the rapid assembly of reusable components. Although you can edit code within Drumbeat 2000, direct programming is not readily supported. A visual programming wizard allows users to generate SQL and ODBC code without programming. When applications are completed, they are moved to an ASP-enabled Web server.

As additional high-level software development tools become available, developers will use them to build secure, enterprise-wide Web applications that apply the power of a database to store, retrieve, and manipulate content.