BRIDGING BOUNDARIES: CORBA in Perspective

SEAN BAKER
Iona Technologies
VINNY CAHILL AND PADDY NIXON
Trinity College, Dublin

Applications that cross the boundaries of different computing machines, operating systems, and programming languages are increasingly the norm. As a result, the need for what might be called "bridging technologies" to develop software that works across heterogeneous environments has become more compelling. The Common Object Request Broker Architecture is one such technology that is both robust and commercially available. CORBA essentially describes how client applications can invoke operations on server objects using the services of an intermediary known as an Object Request Broker, or ORB. CORBA has already been successfully deployed in a variety of application domains ranging from financial information systems to video-on-demand.

While early CORBA adopters were often large commercial enterprises that needed to integrate proprietary back-end applications over diverse operating and hardware systems, CORBA is increasingly finding its way onto the desktop and becoming a serious contender as the technology of choice for the development of Internet-based applications. The standardization of a Java binding for CORBA, supported by products such as Visigenic's VisiBroker for Java* and Iona Technologies' OrbixWeb,* allows client applets to invoke server objects across the Internet.

Consider the example of a ticket agency that implements a CORBA interface to its services. This agency could then use a Java applet from a Web page to invoke one of its servers and book tickets on behalf of a client. Moreover, Netscape's incorporation of an ORB into its Communicator* client software means that many users will already have an ORB on their desktop.

This article introduces CORBA by describing its key components. We then review the boundaries it helps to bridge. We conclude by comparing CORBA with a number of other bridging technologies available today.

TECHNICAL OVERVIEW

CORBA is a standard for interoperability in heterogeneous computing environments. It is controlled by the Object Management Group.* At its core is the specification of

1089-7801/97/$10.00 ©1997 IEEE

the ORB, which facilitates, or *brokers*, the making of requests on objects. CORBA systems are composed of *objects*—components with well-defined interfaces that describe the services they provide to other objects in the system. These objects may be components implemented using an object-oriented language such as C++ or Java, or they may be simple wrappers for large amounts of, possibly legacy, code.

Object Request Broker

The ORB itself normally consists of a library that is linked with each CORBA process and a daemon process that is involved in establishing a connection but not otherwise in communication. Client processes make calls to objects in server processes; but the system is flexible because servers can also make calls on objects (that is, they can behave like clients) and clients can contain objects. The exact nature of an ORB, however, must depend on the environment. For example, an ORB in a real-time environment might dispense with the daemon to save space.

ORBs support other capabilities. For example, the so-called dynamic invocation interface (DII) and dynamic skeleton interface (DSI) shown in Figure 1 allow access to objects without type-specific stubs and skeletons. Moreover, at runtime a client can

- determine the interface to an object for which it has received an object reference;
- make calls to an object without having to know the object's interface at runtime; and
- control the parameters of the ORB itself.

Interface Definition Language

The interfaces between objects are the key to a CORBA system. The ORB must allow one object to use another, even if the two objects are implemented in different programming languages and run on different networked machines with different operating systems. Each CORBA object has an interface defined in the Interface Definition Language. IDL is a simple language with a simple underlying object model; it acts as a lingua franca between different programs and systems. The ORB transports requests on an object to it and performs any required translation between caller and object.

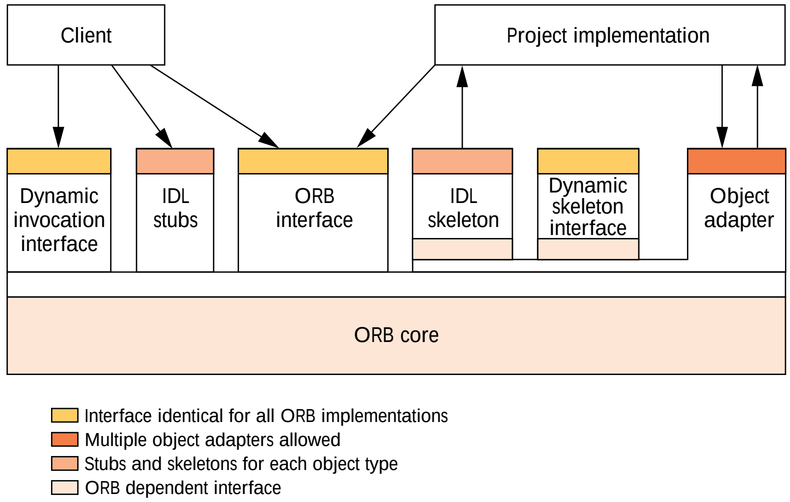

Figure 1. The static and dynamic aspects of the core CORBA standard.

Adapted with permission of the Object Management Group.

The IDL interface is defined by giving a name, possibly an inheritance list, and a list of operations and attributes. Each operation has a name, a return value, possibly some parameters, and perhaps a raises clause. Attributes are a shorthand way to specify a value that can be read and written; however, the returned value can be recalculated as a result of a read request, so the attributes do not represent internal (member) variables.

IDL supports a range of basic types as well as structured types (structs, sequences, arrays, and unions). Because IDL is an object-oriented language, a CORBA system is object-oriented at its global level, even if its objects are implemented in a procedural paradigm. The example in Figure 2 of an IDL-defined interface could be used by the aforementioned ticket agency to define a very simple interface to a cinema.[1]

Any number of objects of type FrontOffice can exist in the same or different servers. Each allows its capacity (number of seats) to be retrieved (but not updated directly because the attribute has been labeled as readonly). The price of a place is returned by a call to the operation getPrice(). This call can be made without concern for what programming language the target FrontOffice object is implemented in, which host it is running on, or what the host's operating system is.

```
struct Place {// a structure containing the row
              // and seat number in that row
    char row;
    unsigned long seat;
};

interface FrontOffice {
    readonly attribute unsigned long numberOfSeats;
    float getPrice (in Place chosenPlace);
};
```

Figure 2. Example IDL-defined interface between a ticket agency and a cinema.

As Figure 1 shows, IDL interface definitions are used to generate client-side stubs and server-side skeletons that provide the link between the underlying ORB and, respectively, the client and server applications. When using CORBA, the client normally invokes a method in a local stub, which forwards the request to the ORB. The ORB is responsible for locating an object that can implement the request and forwarding the request to it before invoking the appropriate method via the skeleton.

IDL lacks any computation capabilities; it is used solely for interface definition. Therefore, every CORBA implementation must map IDL to

- the client programming language in order to make calls to CORBA objects and
- the server language in order to implement, for example, the interface FrontOffice and create objects of that type.

An IDL compiler handles the mapping between IDL and the target programming language, and is therefore used to generate the required stubs and skeletons.

Internet InterORB Protocol

Interoperability among ORBs provided by different vendors is clearly a critical requirement. Thus, OMG also defines the on-the-wire Internet InterORB Protocol that two ORBs from different vendors must be able to use for interoperability over the Internet. A typical IIOP packet encodes the identifier of the target object, the operation being called, and the parameters to be passed to the target object.

While IIOP was not part of the first CORBA specifications, it is now widely supported by ORB vendors. Indeed, stand-alone implementations of IIOP will soon be available. Such implementations can be embedded in other applications, allowing them to, as it were, "speak CORBA" and hence initiate invocations on CORBA objects.

Like IDL, IIOP reflects the attempt to standardize on interoperability, since standardizing on computing environments is so unlikely.

Object Services

CORBA defines a wide range of services to extend the core capabilities of the ORB. Some run as servers on top of the ORB, while others must be built, at least partly, into the ORB. The services range over a wide area of computer science, and each application is likely to use only a few. Among the most important for the bridging functions are

- *Naming*—associates a symbolic name with an object and allows a client to obtain a reference to the object by looking up its name.
- *Event*—allows the decoupling of clients from target objects. The client sends an event (message) to an event channel, and the event is then sent to each of the servers registered on that channel. This allows one-to-many communication.
- *Security*—supports authentication and authorization so that applications can restrict certain or all operations to specified clients; also supports encryption.
- *Transaction*—allows applications that communicate with multiple servers and update multiple databases to atomically commit or abort their changes. Referred to as OTS, this service defines the simple interfaces that clients need to create and terminate transactions. Its other interfaces are internal, and are used by an OTS implementation to abstract the two-phase commit protocol and other details.

OMG maintains complete documentation of CORBA services at http://www.omg.org/corba/csindex.htm.

CORBA also defines a number of facilities, such as the System Management facility, which provides a set of interfaces to help structure the implementation of the management layer. For complete documentation on CORBA facilities, see http://www.omg.org/corba/cfindex.htm.

BOUNDARIES TO INTEROPERABILITY

CORBA was designed to solve the problems of interoperability among the components of (distributed) applications. If we consider different software units (programs, objects, functions, databases, and so on) as components that provide particular functionality, then the problem is to integrate components across different boundaries.

- *Network boundaries*. Distributed applications must address the intricacies of network access and communication, while maintaining network transparency.
- *Language boundaries*. Software producers use a particular language for many different reasons, including skill base, legacy dependencies, and client requirements. Applications written in one language must be able to interoperate with those written in another.
- *Operating system boundaries*. Different operating systems might exist within organizations because of purchasing decisions, client requirements, or the needs of legacy systems. For a variety of reasons, the operating system is seldom a variable that can be changed. Integration between applications that run on different operating systems is a further requirement.
- *Object model boundaries*. Although object-oriented programming has become a de facto standard for systems development, there are still many differences among the object models in use. For example, the object programming models of Java and C++ differ in subtle ways that inhibit integration.
- *Legacy boundaries*. Particularly important from the enterprise perspective is the need to integrate with legacy applications. By mapping

such applications and providing an IDL interface into them, the functionality of these applications can be made available on other platforms.

- *Administrative boundaries*. An organization that provides a service to several third parties and access to this service via a software interface typically ports its access software to each party and maintains these different instances of the system. The administrative overhead and other risks of system changes at third-party sites are another boundary to interoperability.
- *Paradigm boundaries*. Using different design and programming paradigms can generate solutions with component sets that should be implemented in different languages and perhaps based on different sets of assumptions and rules. These differences can be hidden behind an IDL interface.
- *Vendor boundaries*. Being tied to a single vendor can be very restrictive and costly in a business environment that changes daily, and constitutes the final boundary in our list.

Table 1. Boundaries versus bridging technologies.

	CORBA	Java	DCOM	MOM	TP	WWW
Network	Yes	Yes	Yes	Yes	Yes	Yes
Language	Yes	No	Yes	No	No	Yes
Operation system	Yes	Yes	No	No	Yes	Yes
Administration	Yes	Unknown	Yes	Yes	Yes	Yes
Legacy	Yes	Yes	Yes	No	No	Yes
Object model	Yes	No	Unknown	No	No	No
Vendor	Yes	Unknown	No	No	No	Yes
Paradigm	Yes	No	Yes	No	No	Yes

Legend:
Yes = technology can bridge boundary
No = technology does not bridge boundary
Unknown = unable to assess or partial bridge of the boundary

OTHER BRIDGING TECHNOLOGIES

Through its ORBs, IDL, and IIOP, CORBA provides a comprehensive solution to the bridging problem. However, it is not the only solution. In this section, we briefly describe five main alternatives and identify the scope of each. Table 1 summarizes the bridging capabilities described for each alternative. While the picture is likely to change as new offerings become available even in the near term, at this time CORBA offers the most general solution. DCOM appears to be the only other serious contender.

Clearly Table 1 does not show the whole picture. Important issues such as functionality, ease of use, performance, and vendor support, to mention but a few, are not addressed. For example, it seems here that the WWW solution is closely comparable to a CORBA or DCOM solution, but this is true only for quite limited applications. Nonetheless, we hope the table offers a starting point for comparing available middleware technologies.

Java

Java and its companion technologies, especially the JavaBeans component model and the Java Enterprise API (including Java Remote Method Invocation, Java Database Connectivity, Object Serialization, and the Java Naming and Directory Interface), are sometimes seen as a competitor of CORBA. There is certainly some overlap between the capabilities of the two. For example, Java and RMI provide a viable means of building distributed applications—that is, of bridging the network boundary. Since its inception, Java's portability has allowed the operating system boundary to be bridged.

However, at its heart, Java is simply another object-oriented programming language. While CORBA is concerned with the interfaces between objects and application components modeled as objects, Java is primarily concerned with the implementation of those objects. Its suitability as a means of bridging either the language or object model boundaries is, at best, questionable. Similar comments apply to the paradigm boundary. In practice, we expect to see coexistence between CORBA and Java, with each playing to its respective strengths.

DCOM

The Distributed Component Object Model is Microsoft's proprietary object middleware. DCOM has evolved from Microsoft's OLE, COM, and ActiveX technologies. OLE (originally, Object Linking and Embedding) emerged in the late 1980s as Microsoft's initial solution to supporting the now familiar data-centric application model, based on compound documents. OLE 2.0 introduced COM (the Component Object Model) in the early 1990s. OLE 2.0 became simply OLE and eventually represented a wide range of technologies built on top of COM.

COM itself provided a binary interface standard through which services could be packaged (as either dynamic link libraries or applications) in a variety of programming languages. However, COM was substantially limited by its lack of support for distribution. DCOM addressed this deficiency by allowing clients to access services provided across the network on remote machines. OLE technologies have been recently re-branded as ActiveX, and OLE has been given back its original meaning.

DCOM was designed from the outset to provide support for application integration. Support for distribution came later. The character of its object model reflects this history. DCOM supports an object-oriented model, but one that differs substantially from classical object-oriented models. A DCOM object provides services through one or more distinct interfaces. It may implement all of its interfaces itself. Alternatively, it can use a technique called aggregation to delegate one or more interfaces to other DCOM objects. Aggregation allows an application to be constructed from reusable DCOM components.

DCOM also breaks with classical object-oriented approaches in its lack of support for polymorphism. DCOM advocates considered such support inappropriate for a model whose primary purpose is the construction of applications from binary components. Note, however, that DCOM does not place any restrictions on the use of a language supporting polymorphism when implementing a DCOM component.

A client may query a DCOM object (using standard interface methods implemented by all DCOM objects) for a particular interface (identified by a globally unique number known as a UUID). An interface is considered immutable, in that once published, it should never be changed. New functionality is added to a DCOM object only by adding new interfaces.

While there are significant technical differences between CORBA and DCOM, both are concerned with component integration. Thus, DCOM is a viable solution for bridging the network, language, legacy, and paradigm boundaries. Its main limitations stem from the fact that it is not an open standard. Although Microsoft reports that it is working with standards bodies such as IETF and W3C,[5] DCOM is currently available only for Microsoft platforms. This may change as other vendors begin to port DCOM to other platforms in the next few months, but for now, DCOM does not bridge the operating system or vendor boundaries.

Microsoft claims that the release of its new Transaction Server will give DCOM many of the services it has lacked, making it more than simple "plumbing" and into a real enterprise solution that offers naming, event, security, transaction, and life-cycle services. In practice, we expect CORBA and DCOM to coexist and, indeed, the OMG is already considering COM/CORBA interoperability.

World Wide Web

The Web is an example of a set of technologies that supports applications that cross most of the boundaries of interoperability. In its most basic form, the Web allows information supplied via the FORM tag in an HTML document to be sent to a WWW server at which it can be processed by a CGI program. While this approach is extremely flexible, it also has many disadvantages.

The information supplied to the program must be one or more strings that must be converted to appropriately typed values at the server side. In effect, all messages are untyped and structured data types are not easily supported. Sessions are also a problem due to a lack of persistence and the fact that WWW servers normally create a new instance of the CGI program to handle each request. The programming model is strictly client-server, with essentially all processing taking place at the server side. Moreover, this approach obviously applies only to applications that are to be deployed over the Internet. Evans and Rogers[6] provide a more detailed comparison of the relative strengths and weaknesses of using HTML and CGI versus CORBA and Java for the development of a distributed application.

While the HTML/CGI approach will undoubtedly continue to be used, we expect the majority of future Internet applications to be developed using some combination of Java, ActiveX/DCOM, and CORBA.

MOM

Message-Oriented Middleware refers to a collection of proprietary products that enable communication between clients and servers via queues of messages. While providing a basis for developing new distributed applications, MOM systems are proprietary and typically restricted to a small number of platforms.

While MOM systems can differ in their architecture, an interesting variation is one in which several clients send messages via a queue to a set of servers that feed off that queue—with each message being handled by only one of the servers. This gives a natural form of fault resilience and load sharing. CORBA Event Service provides similar functionality except that each of a set of servers receives each event. Both forms of communication are important, and OMG has indicated that the second form will be allowed for in the future.

TP Monitors

Transaction-processing monitors typically provide three services:

- message transmission from clients to servers (but not using high-level interfaces as in CORBA);
- two-phase commit of multiple databases; and
- control over the number of concurrent calls each server must handle.

The core ORB provides the first of these, and the OTS provides the second within the CORBA framework. CORBA does not currently address the third.

TP monitors have been widely used for application integration in enterprise computing but lack the openness and generality of CORBA-based systems. The current trend is toward closer integration between ORBs and TP monitors in the context of the OTS, as exemplified by OrbixOTS from Transarc, which is based on the Encina TP monitor.

CONCLUSION

CORBA is a powerful tool that solves the problem it was designed to solve—namely, interoperability. The development of the Internet as a global computer continues to move the market toward component-based computing. As one of our colleagues is fond of saying, "Heterogeneity is business!" As long as there is heterogeneity, there is a role for CORBA and similar bridging technologies. ■

REFERENCES

1. S. Baker, *CORBA Distributed Objects*, Addison-Wesley, Longman, Harlow, UK, 1997.
2. P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, Mass., 1987.
3. C. McHale, *Synchronisation in Concurrent, Object-Oriented Languages: Expressive Power, Genericity and Inheritance*, doctoral dissertation, Dept. of Computer Science, Trinity College, Dublin, 1994.
4. C. Atkinson, "An Object-Oriented Language for Software Reuse and Distribution," technical report, Dept. of Computing, Imperial College of Science, Technology and Medicine, Univ. of London, 1990.
5. Microsoft Corp., "DCOM: A Business Perspective," http://www.microsoft.com/, posted Aug. 1997.
6. E. Evans and D. Rogers, "Using Java Applets and CORBA for Multi-User Distributed Applications," *IEEE Internet Computing*, Vol. 1, No. 3, May/June 1997, pp. 43-57.

Sean Baker is a co-founder and director of Iona Technologies, which produces Orbix, a leading CORBA-compliant ORB. Prior to establishing Iona, he held a tenured post in the Department of Computer Science at Trinity College, Dublin.

Vinny Cahill is a co-founder of Iona Technologies. He currently leads the Distributed Systems Research Group in the Computer Science Department at Trinity College, Dublin.

Paddy Nixon is a lecturer in the Department of Computer Science at Trinity College, Dublin, and a member of the Distributed Systems Group there.

URLs OF INTEREST

Corba Freeware • www.omg.org/news/freestuff.htm
Corba Information Repository • www.acl.lanl.gov/CORBA/
Corba Introductory • www.infosys.tuwien.ac.at/Research/Corba/ intro.html
Distributed Object Computing Magazine • www.ondoc.com/
Douglas Schmidtís Corba page • siesta.cs.wustl.edu/~schmidt/corba.html
Draft of DCOM Protocol spec and other technical information • www.microsoft.com/oledev/
Java API Overview and Schedule • www.javasoft.com:80/products/api-overview/index.html
Java Documentation • www.javasoft.com:80/ocs/index.html
Java for Developers • www.javasoft.com:80/nav/developer/index.html
OMGs Corba-Related Bookmarks • www.omg.org/news/corbkmk.htm

MAILING LISTS

Corba mail lists • comp.object.corba

Send mail to: majordomo@omg.org with the following command in the body of the message: subscribe comp-object-corba.

DCOM Mail List

Send mail to: Listserv@listserv.msn.com. Subject: (leave blank). Message Text (to subscribe): sub DCOM your name.

JavaCORBA • JavaCORBA@luke.org

Send a mail to listserv@luke.org with SUBSCRIBE or SUBSCRIBE DIGEST in the Subject field.

Java IDL • idl-users@java.sun.com

Send mail to listserv@java.sun.com with the following command in the body of the message: subscribe idl-users.

URLs CITED IN THIS ARTICLE

***OrbixWeb** • www.iona.com/
***Communicator** • www.netscape.com/
***Object Management Group** • www.omg.org/
***Visibroker for Java** • www.visigenic.com/