AN OVERVIEW OF CORBA 3

Jon Siegel

Director, Domain Technology
Object Management Group
492 Old Connecticut Path
Framingham, MA 01701 USA

Siegel@omg.org

Abstract: Already the architecture of choice for distributed enterprise applications, CORBA takes a major step in capability and ease-of-use with the addition of new features labeled, collectively, CORBA 3. The new abilities group into three areas:

Java and Internet Integration: Two URL formats for the CORBA object reference provide access to CORBA services and objects over the internet or on remote hosts in your enterprise. A Java-to-IDL mapping automatically defines IDL interfaces for objects programmed in Java. A binary stub standard removes a possible barrier to portability. And, a Firewall specification enhances enterprise access to CORBA over the internet.

Quality of Service Management: An enhancement to OMG's standard IIOP protocol brings the reliability and flexibility of messaging to your CORBA installation. Asynchronous invocation modes can now be used with stub-based invocations. Quality of Service can be specified for both synchronous and asynchronous invocations, in a number of ways. Minimal CORBA (for embedded systems), realtime CORBA, and fault-tolerant CORBA specifications are either complete or nearly so.

Distributed Components: Perhaps the most exciting of the new developments, the CORBAcomponents specification defines a container which packages the capabilities that enterprise applications rely upon: persistence, transactionality, security, and event handling. Compatible with Enterprise Java Beans, the system extends component technology to C++ and the other CORBA programming languages. Also provided are interface navigation, connection of interfaces supplied and required by the components in an assembly, and a multi-platform software distribution format and installer which enable a CORBA component marketplace.

Keywords: CORBA, CORBA 3, Java, internet, components, quality of service, firewall, asynchronous invocation, scripting, realtime

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

1 INTRODUCTION

1.1 Overview of CORBA and the OMA

OMG's specifications divide into two major parts: CORBA (Figure 1) [1, 2], which provides the object-based interoperability foundation and, built on this foundation, the conceptually layered (but *not* hierarchical) Object Management Architecture (OMA; Figure 2) [3].

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

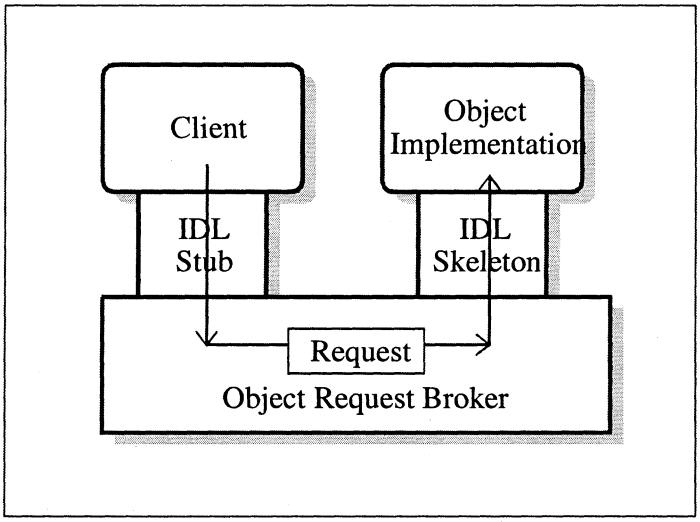

Figure 1. Simplified view of the CORBA architecture.

1. Both client and object implementation are isolated from the Object Request Broker (ORB) by an OMG IDL interface. CORBA requires that every object's interface be expressed in OMG IDL. Clients see only the object's interface; never any implementation detail. This guarantees substitutability of the implementation behind the interface – our plug-and-play component software environment.
2. 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 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 adminstrative burden, concentrates on the problem at hand.

Building on CORBA, OMG's Object Management Architecture (Figure 2) provides the basis for enterprise computing. The CORBAservices provide system-level services needed by almost any object-based system, while the CORBAfacilities, primarily in industry-specific (vertical) areas, allow standards-based access to common datatypes and functionality needed in enterprise computing. Together, they enable an enterprise computing model composed of interoperating objects from multiple vendors and developers – the component software revolution.

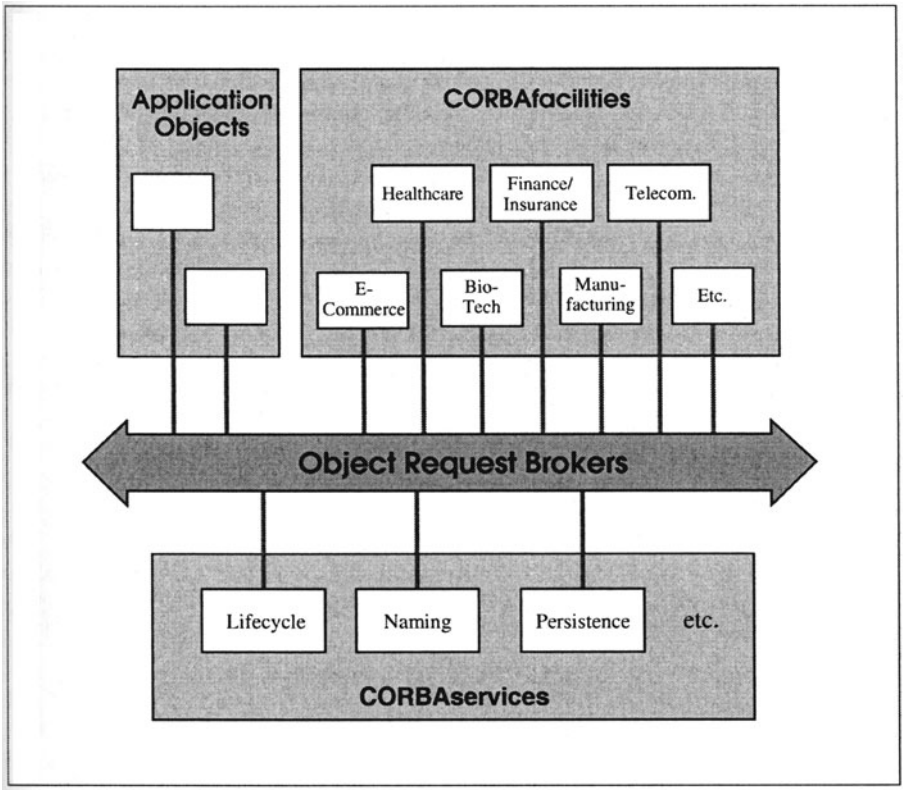

Figure 2. The Object Management Architecture.

1.2 OMG Interface Definition Language

In CORBA, an object's interface is defined in OMG IDL – Interface Definition Language; also an international standard designated ISO/IEC 14750 and ITU-T Rec. X.920. The interface definition specifies the operations the object is prepared to perform, the input and output parameters each requires, and any exceptions which 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 as the object implementation uses to receive and respond. This design provides a great

amount of flexibility, and many benefits. It enforces encapsulation, and allows clients to access object implementations independent of each other's programming language, operating system, hardware platform and data representation, location on the network, native protocol, and other factors.

To the client or user, the OMG IDL interface represents a *promise*: when the client sends a proper invocation to an object through its interface, the expected response will come back. To the object implementor, the interface represents an *obligation*: He must implement, in some programming language, all of the operations specified in the interface. Writing the contract (in OMG IDL), and fulfilling it (in a programming language such as C++, C, or Smalltalk), are usually two separate steps in the writing of a CORBA application, although some vendors' CORBA products generate OMG IDL automatically from either source code or application design information.

For every major programming language, an OMG standard language mapping specifies how OMG IDL types and method invocations convert into language types and function. This is how the OMG IDL skeleton and the object implementation come together: The OMG IDL compiler uses the mapping specifications to generate a set of function calls from the OMG IDL operations. Programmers, probably assisted by an automated or semi-automated tool, refer to the OMG IDL file and use the language mapping to generate the corresponding set of function statements. After compilation and linking, these resolve so that the skeleton makes the right calls to invoke operations on your object implementation. Currently, CORBA specifies OMG IDL language mappings for C, C++, Java, COBOL, Smalltalk, and Ada; a mapping for LISP is being adopted. Mappings don't have to be standardized by OMG in order to be useful; implementations of not-yet-standard mappings are available now for objective C, Visual Basic, Perl, and other languages.

A web of ORB-to-ORB communications pathways provides interoperability among all of the CORBA objects on a network. ORBs use the OMG-specified protocol IIOP (Internet Inter-ORB Protocol) for communication, and the Interoperable Object Reference (IOR) to pass object instance location information among themselves. And, ORBs share OMG IDL interface definitions, which they maintain in their Interface Repositories, to enable data formats to be translated when requests and responses cross system boundaries.

1.3 The Object Management Architecture

OMG's Object Management Architecture (Figure 2) builds upon the CORBA interoperability foundation to realize our vision of plug-and-play component software. A foundation of standard services invoked using standard interfaces, the OMA defines an environment where interoperability penetrates upward from the system level into application components.

The goal of the OMA is simple: when applications provide basic functionality, let them provide it via a standard interface. This enables a component software market both above and below the level of the interface: below it, multiple *interchangeable* implementations of the basic functionality (compound document management, for instance) may still provide differences in performance, price, or adaptation to run on specialized platforms while above it, specialized components (a sophisticated editor

object, for example) come to market which can operate on any compound document managed by a component which conforms to the standard interface.

The CORBAservices specify basic services which almost every object needs; this part of the OMA was started first and the CORBA facilities take advantage of much of it. The CORBA facilities, being specified now, provide intermediate-level services to applications. Application Objects, at the uppermost level, will not be standardized by OMG; this is where vendors compete in innovative ways to provide the best combination of features and value for the customer.

For each component of the OMA, the OMG provides a formal specification – a published document prescribing the syntax (how to invoke each operation on each object) in OMG IDL, and semantics (what each operation does) in English text. Vendors of OMA-compliant systems then implement and sell the services (some bundled with an ORB package, others not), where they are accessed via the specified OMG IDL interfaces. Vendors do not have to provide every service, but every service they provide must conform to the OMG specifications in order to bear the CORBA brand.

1.4 The CORBAservices

The CORBAservices provide fundamental, nearly system-level services to Object Oriented applications and their components. Thirteen CORBAservices have been formally specified so far. They are:

- Lifecycle service
- Relationship Service
- Naming service
- Externalization Service
- Event service
- Object Query Service
- Object Properties service
- Transaction Service
- Concurrency Service
- Licensing Service
- Security service
- Trader service
- Object Collection service.

By now, you should be starting to develop a picture in your mind of how this is all going to work together: CORBA and OMG IDL provide the interoperability infrastructure which our objects will use to link up. Then the OMA standardizes a set of common foundation objects, including the key "matchmaking" services Naming

and Trader, which get clients and object implementations together when they need it, along with other basic services. When a client is ready to use a service or an object, it can find it, it can communicate with it, and it can invoke it.

CORBA 3 *builds* on this basic but capable architecture in three primary areas:

1. Java and Internet Integration;
2. Quality of Service Management; and
3. A Distributed Component architecture and container definition.

2 INTRODUCTION TO CORBA 3

OMG increments *minor* release numbers to signify either the completion of a maintenance cycle, or the addition of minor although significant new capability (such as the Portable Object Adapter or POA), or both. Before CORBA 3, the current version of CORBA was 2.4.

OMG increments *major* release numbers to signify a marked gain in capability. The first increment of the major release number, to CORBA 2, marked the introduction of the IIOP protocol which standardized interoperability of CORBA object request brokers.

CORBA 3, however, consists of a *collection* of new specifications generated by OMG. Several of them, taken alone, would represent significant new capability for CORBA either on the internet, in the enterprise, or in both locations. Taken together, they represent a major gain in capability, ease of use, and suitability for business computing.

The specifications included in the designation CORBA 3 include:

Java and Internet Integration:

- A Java-to-IDL mapping for automatic generation of IDL stubs and skeletons from Java objects [4];
- A Firewall Specification for passage of IIOP protocol through firewalls [5]; and
- An Interoperable Naming Service specification that defines two URL forms for CORBA object references (not officially part of CORBA 3 but relevant here nonetheless) [6].

Quality of Service Control

- A Messaging Specification that not only defines asynchronous and time-independent invocation APIs and semantics for CORBA, but also allows client and object to control Quality of Service parameters [7]; and
- Standards for Minimum CORBA [8] (e.g. for embedded systems), Fault-Tolerant CORBA [9], and Realtime CORBA [10].

CORBAcomponents

- CORBA Objects passable by value [11];
- The CORBAcomponents Specification [12]; and
- A Scripting Language specification [13].

3 JAVA AND INTERNET INTEGRATION

Three specifications enhance CORBA integration with the increasingly popular language Java, and the internet. Of these, two are included CORBA 3; the third, termed the Interoperable Naming Service or INS, is actually a CORBAservice and included in a different collection of OMG specifications. However, its contribution to CORBA-internet integration forces us to include it here.

A cardinal principal of CORBA architecture is programming-language independence. Java's object orientation and architecture make it particularly suitable for use with CORBA, and enable the reverse language mapping. However, this does *not* signal any change of philosophy for CORBA which remains steadfastly language-independent. (Remember this when a new "perfect" language replaces Java a few years from now.)

3.1 Java-to-IDL Mapping

OMG currently defines standard mappings from IDL to six programming languages: C, C++, Java, COBOL, Smalltalk, and Ada [12]. For any legal IDL interface, these mappings define the program-language types and constructs that correspond, and enable interoperable clients and object implementations to be built in any of these languages.

Besides being object-oriented, Java is inherently distributed via the Java RMI or Remote Method Invocation. The standard IDL to Java mapping defines an API that allows Java clients to invoke operations on CORBA objects. A reverse mapping defines an API that allows CORBA clients – regardless of the language they are written in – to invoke Java objects.

The reverse mapping starts by defining the subset of Java that conforms to CORBA sufficiently to support the reverse mapping; very little, primarily special cases, is excluded. The mapping pays particular attention to Java RMI/IDL value types, whose values are moved between systems rather than references being passed. The new OMG IDL *valuetype* supports this feature. Also treated are inheritance considerations and exceptions. Attention to ordering of parameters assures that parameters occur in the same order on the wire in Java RMI invocations and IIOP invocations of the reverse-mapped interface.

At the object implementation end of a connection, the Java Virtual Machine (JVM) uses the reverse mapping implicitly, to construct a CORBA interface for its running objects, and to define the IIOP protocol messages that it accepts and emits. To construct a client, it is necessary to run the Java code through a reverse-compiler to generate the IDL explicitly. This is may be compiled into a client stub in *any* CORBA-supported programming language. The client is then written, compiled, and

run with the stub using an ORB that speaks the OMG-standard protocol IIOP, to invoke operations on the Java object.

The result is an environment where Java RMI objects are available to all CORBA clients, without the Java programmers having to learn CORBA or any distribution method besides RMI. This helps integrate Java into the multi-language CORBA environment.

The mapping is *not* the reverse of the IDL to Java mapping, and a round-trip does not yield the original code. This is explicitly not a goal of the mapping.

3.2 Firewall Specification

Businesses protect their enterprise networks from attack by outsiders with firewalls. CORBA, with its location-transparent execution capability, looks like a security threat to a firewall which may trap IIOP protocol and prevent CORBA invocations across its barrier. The Firewall Specification allows these invocations without removing the protection that the system was designed to effect.

The specification defines mechanisms for dealing with three types of firewalls:

- **TCP Firewalls:** This is the first example of a *transport-level* firewall; these perform access control decisions based on address information in TCP headers, typically host and port numbers. To facilitate this, IIOP requires a "well-known port number" assignment from the Internet Numbers Assignment Authority or IANA. These ports have been assigned; the well-known port for IIOP is 683, and for IIOP/SSL it is 684.

- **SOCKS Firewalls:** The SOCKS protocol defines a proxy that serves as a data channel between a client and server communicating over either TCP or UDP. Following authentication of the client to the SOCKS proxy server, the client requests and the proxy server connects to the requested real server and the client starts passing data to it.

  SOCKS is simple to implement for ORB vendors, who only have to re-link their products with a SOCKS-ified TCP library. All differences between simple TCP and SOCKS TCP are taken care of by the library; APIs are identical. Requests are sent to the SOCKS proxy server instead of the target CORBA server; the proxy server in turn routes them to their destination.

- **GIOP Proxy Firewalls:** This is a new Firewall type defined by the specification. Unlike the first two, it is an application-level firewall that understands the GIOP messages and CORBA headers. Because of this, it is the only one of the three that can perform object-level filtering. The specification also defines a *pass-through* connection that does not examine invocation data that may be, for example, encrypted and therefore opaque to the firewall, but even this type of connection would be object-specific and therefore more precise than either the TCP or SOCKS firewalls could provide.

In CORBA, objects frequently need to *call back* to the client that invoked them; for is, 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 the establishing of a second TCP connection for this traffic heading in the other direction. One way firewalls protect is by not allowing incoming connections, which also prevents callbacks that are necessary for applications to work. Since clients register with objects for callbacks that may be important ("If the prime rate falls .25%, let me know so I can sell all my stock"), the firewall specification defines a way for this to work.

Under the new specification, an IIOP connection is allowed to carry invocations in the reverse direction *if* the object reference for the target of the reverse invocation was sent to the remote server (now acting as client) over that same connection. ORBs at both ends must be enabled for bi-directional IIOP, and must keep track of object references that have been sent over it. (The Interface Repository provides infrastructure enabling for this.) So, when the client wants to be called back, it instantiates a callback object and sends its object reference to the remote server over its established IIOP TCP connection. The remote server sets up a client which, when it is triggered, invokes the callback object over the same connection in the reverse direction.

3.3 Interoperable Naming Service

OMG's platform specifications fall under the umbrella of CORBA – the Common Object Request Broker Architecture. Supporting CORBA is the Object Management Architecture, which includes the CORBAservices and the (primarily domain or vertical-market oriented) CORBAfacilities. The Interoperable Naming Service is a CORBAservice and thus not part of CORBA 3, but it relates so closely to internet integration that we will mention it here.

The CORBA object reference is a cornerstone of the architecture: When an instance is first created, its ORB (and, for recent products, its POA) creates a standard-format Interoperable Object Reference (IOR) that contains all of the information a remote ORB needs to invoke the instance. Because 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 for even that?

The Interoperable Naming Service defines one URL-format object reference, *iioploc*, that can be typed into a program to reach defined services at a remote location, including the Naming Service. A second URL format, *iiopname*, actually invokes the remote Naming Service using the name that the user appends to the URL, and invokes the named object. As an example, the URL

iioploc://www.omg.org/NameService

would resolve to the Naming Service running on the machine whose IP address corresponded to the domain name www.omg.org.

4 QUALITY OF SERVICE CONTROL

4.1 Asynchronous Messaging and 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 may be retrieved by either polling or callback, with the choice made by the form used for the original invocation.

Policies allow control of Quality of Service of invocations. Clients and objects may control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live; set a start time and end time for time-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.

4.2 Minimum, Fault-Tolerant, and Real-Time CORBA

Minimum CORBA, primarily intended for embedded systems, has just been defined by OMG. Embedded systems, once they are finalized and burned into chips for production, 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.

Thus, the Minimum CORBA specification removes these as well as features for dynamically creating, activating, and passivating objects and serving requests. In order to preserve interoperability, all of OMG IDL is included, including the Any type. The specification points out that, although compliant ORB products have to include these features so they are available at code and link time, unused features may be omitted from the final executable since selective linking is out of the scope of OMG specifications.

Real-time CORBA is being defined by OMG as this paper is being written. Although the specification is not complete enough to include here, we do know that it will include two scheduling modes: fixed-priority scheduling, and an alternative combination of priority-based scheduling and resource control. For end-to-end predictability, it will specify ways to control ORB resources.

Fault-tolerance for CORBA is being addressed by an RFP, also in process, for a standard based on entity redundancy, and fault management control.

Real-time and fault-tolerant CORBA, whose specifications are farther behind than any of the others listed here, likely will not be part of CORBA 3 when it is first defined. However, their effect on Quality of Service control means that they will be added to the specification as soon as they are ready.

5 CORBACOMPONENTS PACKAGE

5.1 CORBA Objects Passable by Value

Termed *valuetypes*, CORBA objects passable by value add a new dimension to the architecture. Like conventional CORBA objects, these entities 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. Because valuetypes may be input, output, or return values on an invocation, the sending or receiving context is not restricted to either client or object end of the invocation.

Frequently used to represent nodes in binary trees or cyclically-linked lists, valuetypes have been specified and implemented to faithfully represent these important constructs. When the root node of a binary tree is transmitted as a valuetype, its state is transmitted as part of the package. Of course its state includes the next two nodes, so they go too, along with the nodes they refer to and so on through the remainder of the tree. At the end of the process, the entire tree has been transferred and reconstructed in the receiving context, with each node now pointing to transferred nodes at the new site. The specification requires that cyclic structures transferred as valuetypes also reconstruct properly.

5.2 CORBAcomponents

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

- A container environment that packages transactionality, security, and persistence, and provides interface and event resolution;
- Integration with Enterprise JavaBeans; and
- A software distribution format that enables a CORBAcomponent marketplace.

The CORBAcomponents container environment is persistent, transactional, and secure. For the programmer, these functions are pre-packaged and provided at a higher level of abstraction than the CORBAservices provide. This leverages the skills of business programmers, who are not necessarily skilled at building transactional or secure applications, who can now use their talents to produce business applications that attain the necessary attributes of transactionality and security automatically. At run-time, a system administrator installs a CORBAcomponents ORB and configures it to run in a transactional and secure mode, and connects it to a persistent store. Components that are installed in the container automatically become transactional, persistent, and secure.

Containers keep track of event types emitted and consumed by components, and provide event channels to carry events. The containers also keep track of interfaces provided and required by the components they contain, and connect one to another where they fit. A set of interfaces for interface navigation is defined; CORBAcomponents automatically gain this functionality from the ORB when they are built: clients can navigate from any component interface to the master component

interface, where they can discover all of the interfaces available from that component.

Enterprise JavaBeans (EJBs) will be CORBAcomponents, and can be installed in a CORBAcomponents container. Unlike EBJs, of course, CORBAcomponents can be written in multiple languages.

The specification also defines a multi-platform software distribution format. The platform-independence of CORBA has, perhaps, hampered the development of a marketplace in CORBA objects because the distribution of multiple executables lacks economy of scale. To overcome this, the container includes a standard installer that, when activated, pulls the correct executable of a CORBAcomponent from a multi-platform CD-ROM (or other distribution medium) and installs it. The installer program also pulls configuration files from the medium, allows the installer to finalize the configuration via a GUI-based tool, and installs these as well. In a final step, the installer connects event channels and provided and required interfaces.

5.3 Scripting Language Specification

Component assembly becomes a viable programming mode where a component environment and container are present, and a number of configurable components are available in the marketplace. The preferred mode of programming for component assembly is through scripting languages. Unlike the usual programming languages, scripting languages are not useful for detail work but make it easy to define how components fit together to solve business problems. Typically running in interpreted mode without a compilation step, these languages make it practical for business users to assemble components into useful applications.

Because the scripting language specification was not complete when this article was written, we cannot review the exact specification here. However, OMG members expect that the specification will describe a general mapping from IDL to scripting language constructs (modules, types, exceptions, invocations), and be accompanied by mappings for a number of specific scripting languages. Invocations of the ORB pseudo-object will probably route through static mappings, while the dynamic invocation interface and dynamic skeleton interface provide access for application-specific interfaces.

6 SUMMARY AND CONCLUSIONS

After ten 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 CORBA installations.

Although the valuetype underlies many of the CORBA 3 extensions, it has proven difficult for vendors to implement. Valuetype mappings for Java and C++ were produced quickly, but those for the other languages – C, Ada, COBOL, and Smalltalk – lag behind although work has started on most of them. Users of these other languages will not be able to participate fully in CORBA until the mappings and products are complete. The CORBAcomponents specification is a major work in itself, and (following the usual process whereby submitters of OMG specifications have up to a year to produce and market a product) implementations will not be

available for months – perhaps up to a year – after the specification is adopted and this is not expected until March or May of 1999.

However, all expect the result to be worthwhile. Although IDL and the CORBAservices make CORBA accessible to programmers now, they represent a barrier to business users who want to manipulate objects that look just like business entities. CORBAcomponents and scripting will soon enable these business users to assemble applications tailored precisely to their needs, while asynchronous invocation interfaces and Quality of Service control allow sites to take good advantage of networked facilities even where resources are stressed.

References

[1] OBJECT MANAGEMENT GROUP, http://www.omg.org/library/c2indx.html
[2] SIEGEL, J., CORBA 3 Fundamentals and Programming, Wiley and Sons, New York, 1999.
[3] OBJECT MANAGEMENT GROUP, http://www.omg.org/library/csindx.html
[4] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-04-04.pdf
[5] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-05-04.pdf
[6] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-10-11.pdf
[7] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-05-05.pdf
[8] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-08-04.pdf
[9] OBJECT MANAGEMENT GROUP, http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html
[10] OBJECT MANAGEMENT GROUP, http://www.omg.org/techprocess/meetings/schedule/Realtime_CORBA_1.0_RFP.html
[11] OBJECT MANAGEMENT GROUP, ftp://ftp.omg.org/pub/docs/orbos/98-01-18.pdf
[12] OBJECT MANAGEMENT GROUP, http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html
[13] OBJECT MANAGEMENT GROUP, http://www.omg.org/techprocess/meetings/schedule/CORBA_Scripting_Language_RFP.html

Biography

Dr. Jon Siegel, Director of Domain Technology, chairs OMG's Domain Technology Committee responsible for specifications in vertical markets including Finance, Electronic Commerce, Healthcare, Telecommunications, Manufacturing, Transportation, and Business Objects. He is the author of numerous magazine articles, and principal author and editor of the book, "CORBA Fundamentals and Programming" and its second edition, "CORBA 3 Fundamentals and Programming". As a spokesperson for OMG, Siegel is a frequent speaker at conferences, meetings, and private corporate briefings around the United States and the world.

Dr. Siegel comes to OMG after twelve years with Shell Development Company, the research arm of Shell Oil, where his most recent position was in the Computer Science Research Department. Siegel's background includes extensive experience in distributed computing, object-oriented software development, and geophysical computing, as well as theoretical and computational work at the Argonne National Laboratory. While still at Shell, he served as that company's end-user representative

to OMG. For the last year, he also played an active role in several OMG subgroups, chairing the Life Cycle Services Evaluation working group and the End User SIG and serving on the Object Services Task Force. In addition, Siegel served as Shell's liaison to the Open Software Foundation, and served on the OSF End User Steering Committee. He holds a doctoral degree in Theoretical Physical Chemistry from Boston University.