Articles & White Papers

Inside Corba Services

Object Management Group's standard Corba services find way into products, easing the I/S task of building distributed applications

by Max Dolgicer Author's note

Until recently, many corporations were faced with an important dilemma -- wait in frustration for mature Common Object Request Broker Architecture (Corba) technology or have the internal I/S operation build the services. Some corporations decided to wait, and the consequences of such a decision questions the usefulness of Corba as many observers believe only applications with limited complexity can be developed by using the basic Corba core (or a standard ORB). Other corporations decided to develop their own services and quickly found the I/S unit in the midst of hard-core software development efforts, often forgetting that insurance companies or investment banks, for example, should not be in the software development business.

A quick review of Corba architecture

Corba and Corba products are often referred to as a class of software called middleware. Middleware encompasses a diverse category of products, including Message-Oriented Middleware (MOM), the Distributed Computing Environment (DCE), transaction processing monitors, Corba products and Microsoft's OLE/COM technology. The first three categories are referred to as non-object-oriented middleware, or procedural middleware. Each of the categories of products, especially MOM and

TP monitors, have already penetrated multiple corporate environments. The problem with non-object-oriented middleware is that each of those product categories offer developers a relatively low-level API. The APIs are too low-level and do not allow application developers to naturally express and expose any business vocabulary. Business activities such as "flight reservation" or "bank deposit" cannot be referred to within those APIs. On the contrary, object-oriented middleware can provide corporate developers with high-level APIs capable of exposing and expressing business activities as objects, thus "objectifying" the business. This feature alone is very attractive. It is no wonder that many organizations have already adopted, or are in the process of adopting, object-oriented strategies. They are looking for middleware that provides a higher level of abstraction in order to match OO tools and OO programming languages. Such middleware is represented by Microsoft's OLE/COM and a number of Corba-compliant products.

The Common Object Request Broker Architecture (Corba) is a set of specifications published by the Object Management Group (OMG), Framingham, Mass., an industry consortium charged with defining a set of interfaces for interoperable object oriented software. The OMG has defined an object management architecture (OMA) that has become the industry standard reference model for object technology. The OMA identifies four categories of software interfaces: The Corba Object Request Broker (ORB); Corba services; Corba facilities and Corba domains. **(See Fig. 1.)**

Click to link

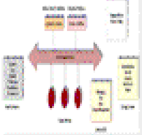

Figure 1

In the heart of the Corba architecture is the Object Request Broker and interface definition language (IDL). (See major Corba components below.)

Corba domain interfaces represent vertical market areas, such as banking, airlines, telecommunications and health-care. Such interfaces are the foundation of industry specific business objects. Corba facilities include higher level generic horizontal interfaces that can be shared across different Corba domains. Examples of these interfaces are: systems management; task management; information management (such as compound documents); and data interchange. The fourth component of OMA is Corba services, which focus on abstractions of the classic middleware services essential for building any serious enterprisewide, distributed application. In addition to the OMA, the OMG defined a standard Corba architecture. **(See Fig. 2.)**

Click to link

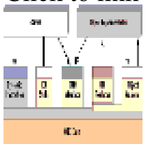

Figure 2

The major components of the Corba architecture are:

OBJECT REQUEST BROKER (ORB) -- An ORB is a message switch or a middleman that allows client-side objects to communicate with server-side objects. It works in a way similar to Remote Procedure Calls (RPC). According to the Corba specifications, all the initial communication that happens between objects is accomplished via the ORB. Consequently, an object is transparent whether it is local or remote. How an ORB is actually implemented is not defined in Corba specifications, and thus there are as many different ORB implementations as there are products on the market. Like RPCs, a key component of ORB technology is an IDL.

INTERFACE DEFINITION LANGUAGE (IDL) -- The Interface Definition Language is the specification language that documents the interface contracts between the clients and servers. However, in the case of Corba ORBs, IDL has to be evolved to support the connects that are part of OO programming (inheritance, for example). In short, IDL is a notational language used to describe the interfaces of objects. Using IDL, methods (or functionality), such as "bank deposit" and their interfaces, can be expressed by the developer in a standard way.

INTERFACE REPOSITORY -- The Interface Repository contains all IDL definitions that describe the attributes, operations, user-defined types and exceptions of the server objects.

BASIC OBJECT ADAPTER (BOA) -- The BOA represents the interface between the ORB and server applications. It dispatches objects that the server application maintains and exchanges messages with the server objects.

STATIC INVOCATION INTERFACE (SII) -- The SII is a stub-based interface used by client programs in order to invoke services on application objects.

DYNAMIC INVOCATION INTERFACE (DII) -- The DII is a generic interface that does not require stubs, but rather supports dynamic construction of object invocations by the client program at runtime.

OBJECT REFERENCE -- The Object Reference is the value that unambiguously identifies an object. Object references are never reused to identify another object.

The original Corba specifications have been extended with the Corba 2 specifications. Among other enhancements, Corba 2 addresses the issue of interoperability between ORBs of different suppliers. A lightweight wire protocol called Internet Inter ORB Protocol, (or in short IIOP), has been defined. As a result, all Corba 2-compliant products should provide interoperability between each other using IIOP protocol.

In sum, Corba specifications define how objects are defined, created, dispatched, invoked, and most important, how they communicate with each other. In real life, the specifications have been implemented by a number of products that are referred to as

Corba products or ORBs. In fact, many suppliers today have implemented basic Corba specifications, sometimes referred to as the Corba core. The basic implementation of Corba specifications, including the ORBs, IDL, SII, DII, Interface Repository and other basic elements required by OMG, is quickly becoming a commodity and all Corba vendors are coming under the increased price pressure for basic Corba products. Enterprises are legitimately questioning how much the basic ORB is worth to developers, especially when Microsoft's ORB (COM/DCOM) is offered for free in a bundle with the Windows NT operating system. In our opinion however, the real question for I/S units is not the price, but whether the functionality offered by a basic ORB is robust enough to develop serious mission-critical distributed applications. In our opinion, the answer is NO. Therefore, a litany of middleware services, whether already defined by OMG or not, are becoming very handy.

What do these services do, and why are they so important to the development of serious distributed applications? Before we address this question, another question comes to mind. Given the fact that basic Corba functionality has quickly become a commodity offered by many Corba firms, how then can suppliers differentiate themselves? This is where support for middleware services is taking the center stage.

Middleware services -- the Corba services

In addition to the basic middleware functionality defined in Corba and offered by different ORB products, OMG has adopted and continues to adapt a number of value-added functions represented by middleware services called Corba services. Some of these services are essential for development and deployment of enterprise applications, and some are less important. Further, some vendors offer functionality not yet defined by OMG. Some services already defined by OMG are supported by suppliers, but in a proprietary (non-Corba-compliant) fashion.

What are these services and how important are they? Overall, Corba services can be divided into two categories, the "must haves" and the "good to haves." Some observers divide Corba services into mortal and non-mortal services. Services differ widely by importance to developers and the level of complexity posed to vendors that need to implement the services. We’ll start from what are the mortal services.

In all cases, the OMG defines only the interfaces for services, not the implementation, and support for all Corba services is not mandatory. Therefore, suppliers have to decide whether to implement a service in question and more importantly, how to implement such a service. Some suppliers develop services in-house while others opt to develop only interfaces to existing services support and decide to leverage existing technologies. In the case of a naming service, for example, a vendor can develop only the Corba interface to the Naming Service while the implementation underneath could be an existing commercial naming services, such as X.500, NDS or DCE. **(See Fig. 3.)** Generally, distributed environments pose challenges and force us to answer questions that are irrelevant in monolithic mainframe-like environments. Such questions include: Where is my object? How do I find it in a distributed environment? The answer is found in the Naming Service.

Click to link

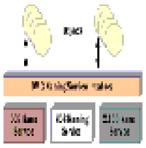

Figure 3

The Naming Service provides the ability to find an object by name, to refer to objects with digestible names and to bind object names to object references within context. Of course, it also provides the name to address resolution. According to OMG specifications, a naming service is simply a well-known repository that stores named object references. The key benefit of the OMG Naming Service is its distributed capabilities, and that it allows for stored object references to be accessed through a Corba environment. Clients can use the Naming Service to locate objects in a Corba environment. Servers advertise themselves with the Naming Service by providing an object reference and an associated name.

In general, any distributed middleware infrastructure needs a comprehensive Naming Service. In fact, a Naming Service should be one of the most essential services provided by any middleware. It is essential to achieve location transparency, an increasingly important requirement as more truly distributed applications are developed and put into production. Corba requires a Naming Service just like any type of middleware. A number of Corba products have already implemented Naming Services. Today, the implementation of Naming Services varies among Corba suppliers from very primitive to adequate. For example, most suppliers have yet to implement name space replication or federated naming (where several naming domains can cooperate). Some vendors have implemented proprietary but excellent naming that does not comply with OMG specifications. For example, SuiteDome from Suite Software, Anaheim, Calif., is one of the very few middleware offerings that has implemented a fully replicated and federated name space. However, good Naming Services implementations can be found in other middleware categories. Corba products could benefit by attempting to leverage on the existing Naming Services provided either by other middleware products or standalone offerings.

In sum, the OMG Naming Service provides the basic name to address resolution among other functions. The ultimate location of the desired object is resolved as a direct result of a name space lookup. In Corba, the Naming

Service does not support complex search criteria capable of locating objects based on attributes and properties. This is the role of the Trader Service, which supports advertising and discovery of services at runtime.

Traditional Corba architecture revisited

From a communication point of view, Corba allows for synchronous and deferred synchronous requests. With synchronous requests, the client application is blocked until the request is returned to the client. With a deferred synchronous request, no blocking occurs. A client application that issued the request is not blocked and is free to do whatever it likes. However, it must periodically poll the ORB for a response from the object which executes the request. There is no provision in the Corba core for features such as asynchronous event-driven communication. In short, the Corba core addresses only requests where the network connection between the client object and the server object must be maintained. This type of communication can support only tightly coupled clients and servers. This does not necessarily reflect the reality of the business activities which often evolve unpredictably and asynchronously. To be more viable, the Corba architecture should support connectionless and event-driven communication. It should have an ability to de-couple among client objects and server objects. This is exactly what an Event Service provides.

The Event Service provides an application architecture where there are no clients or servers. There are suppliers (or generators) of information and consumers of information. The Event Service de-couples communication from specific clients-servers pairs. Using the Event Service vocabulary, suppliers and consumers are de-coupled from each other. The heart of OMG's Event Service contains an Event Channel. **(See Fig. 4.)** The Event

Channel is an object that does the actual de-coupling between suppliers and consumers. Alone, the Event Channel is the consumer to the suppliers, and the supplier to the consumers. The Event Channel accepts connections from one or many suppliers, and one or many consumers. The key is that any event received from one of the suppliers is transmitted to every consumer. Furthermore, multiple Event Channels are supported working independently of each other. An event is defined as any piece of data that has been generated as a result of some activity.

Click to link

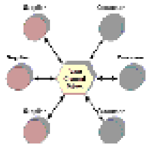

Figure 4

Event Service supports two models of operation -- a push model and a pull model. Both models govern how suppliers communicate with an Event Channel, and how an Event Channel communicates with consumers. In a push model, a push supplier sends an event to the Event Channel using a Corba push operation. This is unsolicited, event-driven processing to the Event Channel. However, the Event Channel also supports a pull model, acting as a client to the suppliers, polling the suppliers for information. Such pulling can be based on time intervals. On the consumer side, both push and pull models are supported by the specification.

Communication between each supplier and the Event Channel, and between each consumer and the Event Channel consumes a network connection. Thus there are clear limits to scalability and to how many concurrent connections each Event Channel can handle. In fact, when an event from a supplier needs to be distributed to many consumers, an Event Channel will send it for each consumer connected to it. Such operations are not efficient and can consume substantial network bandwidth.

The Event Service specification also raises additional questions. What happens to the event if the connection between the supplier and the Event Channel, or between the Event Channel and the consumer, fails in the middle of transmission? Can one-time, and only one-time, delivery of an event be guaranteed? When the consumer receives an event from an Event Channel, can this operation be synchronized in one transaction with the database operation performed by the same consumer? If an Event Channel has the capacity to hold an event's information, how long should it be held and how, and who will control it? Are all events equal and is delivery indiscriminate, or do we discriminate events by a priority that governs the order that events will be sent to the ultimate consumers?

The answer to at least some of these questions lies in what is called Quality of Service, or in short QoS, which is not addressed in the Event Service. These questions will be addressed in an Asynchronous Messaging Service, which has yet to be adopted by the OMG.

In sum, Event Service is important and capable of extending Corba's otherwise synchronous communication model. By implementing an Event Service, Corba suppliers make products applicable to a broader constituency of application developers.

The bread and butter of business applications require transactional integrity. For example, in a case of funds withdrawal, a financial application generally requires that at least two databases be updated if the operation succeeds, or that none of the databases be touched if the operation fails. If a transaction is interrupted before completion, partial updates must be rolled back. Event Service does not provide support for such capabilities. Object Transactional Service (OTS) does.

The Object Transactional Service extends transactional semantics to distributed, object-oriented applications. Application developers should not be concerned that the outcome of any transaction will NOT be deterministic. They can be assured that no matter what happens, each transaction will be a success or failure. Business transactions such as an ATM withdrawal will either credit one account and debit another as one whole, or neither of the accounts will be affected. By the same token it can assure that an airline reservation for two people which are a couple, will be processed as one transaction with a clear outcome -- either both people or neither will have a confirmed reservation.

OTS is one of the most impressive services defined by OMG. The OTS specification is flexible, complete and comprehensive. During the process of specification development, special care was taken to support interoperability between object-oriented and procedural transactional applications when necessary. From the onset, OTS was designed to accommodate the fact that single transactions can include ORB and non-ORB-based applications, which can update object-oriented and non-object-oriented resources. Also, interoperability between the OTS model and the X/Open model defined by The Open Group (formerly called X/Open) is provided. Objects can access legacy (non-object) programs and resources, and by the same token procedural programs can access objects. These developments should assure an evolutionary rather than a revolutionary approach to developing new object-oriented transactional applications. The ability to wrap and integrate with existing TP systems is a key feature of OTS. Finally, OTS supports two transactional models -- flat and nested transactions. Most existing TP applications are built on the flat transactional model where each transaction has a single start and end at a single level. On the contrary, nested transactions support a transaction hierarchy where there is a root transaction (or top level), and subtransactions called children. Transaction nesting can be extremely sophisticated and complex but the bottom line is that OTS is capable of enforcing transactionality across nested transactions without application programmers writing a single line of code. The root transaction cannot commit unless all of its subtransactions have completed. Obviously, when a root transaction is rolled back, all of its children are rolled back as well.

OMG did it and now it is up to Corba developers to implement the service. A number of Corba products are now being extended with transactional capabilities. For example, Iona

Technologies, Dublin, Ireland, is close to shipping two Object Transaction Services as part of the Orbix OTM product that was developer with The Pittsburgh, Pa.-based Transarc unit of IBM and Paris-based Groupe Bull.

The implementation of OTS by Corba products will substantially extend functionality. For example, OTS will enable business applications developed with Corba to update multiple heterogeneous databases in a single transaction. However, implementation of OTS will not automatically provide Corba products with additional features offered by Distributed TP Monitors. Features like failover, load balancing and security, are mostly still missing. There are some exceptions. SuiteDome supports failover and provides bulletproof security while Visigenic Visibroker support redundancy. We expect to see these features implemented in more Corba products.

In sum, implementation of OTS by Corba vendors is very important. It provides the foundation for a new breed of middleware, called Object Transaction Middleware, or in short OTM. One OTM, Microsoft's Transaction Servers (formerly called Viper), is already shipping. However, this is a subject to another article that we will be addressing in the near future. What are the other mortal services?

Additional Corba and non-Corba services -- the present and
the future?

So far we have discussed the four services, Naming, Events, Transactions and Security. These are often referred to as mortal services. It is hard to see a serious application running in production without supporting at least a few of these services unless corporate developers build an equivalent function in-house. We also discussed the Trading Service for its direct relationship to the Naming Service.

However, mortal services represent only part of the middleware services defined by OMG. There are more services. Some are very important while others represent specialized capabilities that are useful only in very specific applications. A short description of some additional services follows:

- Concurrency Control Service: Enables multiple clients to coordinate access to shared resources like databases, files, documents and the like.
- Relationships Service: Provides the capability to represent entities and relationships.
- Externalization Service: Defines protocols for externalizing and internalizing objects.
- Life-Cycle Service: Defines conventions for creating, deleting, copying and moving objects.
- Persistence Service: Provides a set of common interfaces for storing the state of an object.

There are also a number of services -- asynchronous messaging service, licensing, change management and time management -- that are in the process adoption by OMG.

Today, support for Corba services varies widely among different Corba vendors. Of course, products that support mortal services like security, naming, events and transactions, are better positioned to become enterprise middleware solutions. It remains to be seen which other Corba services gain widespread support. Obviously, general services such as Life Cycle will be more important than other more specialized services. In the end, support (or lack of) for middleware services is what will set different Corba vendors apart, which begs the question of how important it is to implement Corba services compliant with OMG specifications. As one customer put it: "Although standards might represent the least common denominator, they have the tendency to change slowly which provides us with some protection of investment in this turbulent world of middleware." At this point, it is hard to say whether this customer represents a majority opinion. As a consulting company, we concur that at the final account, what matters is the robustness of the middleware service implementation. The functionality and the implementation of a service is probably more important than full compliance with Corba service specifications.

Are Corba services sufficient?

Even if all OMG Corba services are implemented, it may not be sufficient to build an infrastructure that can support a multitude of new applications. In particular, if we consider large-scale applications that need to support a substantial number of nodes across a large geographically dispersed network that requires a 7X24X365 availability and split second response time. More required services include:

- Load balancing;
- Fault tolerance;
- Asynchronous communication between objects that supports diverse Quality of Service (QoS) including guaranteed delivery of requests and replies; and
- Multicast capable of supporting many-to-many interactions.

While there is no question that these services are important, they have not yet been defined by the OMG nor have they been incorporated into most Corba products. However, many of the services can be found in non-Corba products and, with proper integration in place, can be leveraged by Corba products. This brings up another subject -- the power of middleware combinations.

Middleware combinations are powerful -- integrating Corba with Message-Oriented Middleware

Support for diverse styles of communication models (synchronous, asynchronous and publish/subscribe), event-driven processing and high scalability are some of the strong points of a number of MOM products. With the flexible QoS, including built-in support for features such as guaranteed message delivery, message expiration (what is called time-to-live), different modes of confirmation, some MOM products are very attractive. Integrating Corba with MOM products can present a very powerful story. While preserving the benefits of Corba products, such an integration can offer Corba functionality found today only in non-Corba products. The forthcoming integration of Iona’s Orbix with IBM’s MQSeries and Expersoft Powerbroker with Modulus Technology’s InterAgent toolkit is a good example.

When integrated, Orbix application developers will be able to take a full advantage of functionality offered by MQSeries (different Quality of Service including guaranteed message delivery, message expiration, confirmations on message arrival) while preserving the relative simplicity and high-level of programming abstractions offered by Orbix. We expect more such integrations to take place.

Conclusion

Corba products are quickly maturing as more suppliers start implementing new middleware services for developing enterprise-scale distributed applications. However, some services are still missing. Middleware combinations have a strong chance of producing "best-of-breed" solutions today. Such solutions could fill the gap found in some individual products. Alternatively, I/S departments can either fill the missing functionality gap by developing additional services in-house or wait for vendors to deliver THE solution. You have to draw your own conclusion based on your unique set of circumstances. After all each company is unique.

[ADT Main | Table of Contents | SPG Main]

---

(508) 652-1100   FAX (508) 652-1220   e-mail:general@spgnet.com http://www.spgnet.com