Articles & White Papers

Object-Oriented and Component-Based Development

**Selecting a distributed object infrastructure is hard;
here is a guide through the maze of vendor claims and product categories**

The Object Infrastructure Challenge

With slogans such as "Distributed Objects Everywhere" and "Orbix Everywhere" being widely repeated, there is no question that object technology is hot. Like it or not, acronyms like Corba, OLE, COM and OpenDoc are becoming part of our daily vocabulary.

Distributed object computing may substantially change how we develop, deploy and maintain applications. Distributed objects may make obsolete all we have learned over the past 20 years. In short, it is possible that distributed objects are the panacea.

As consultants specializing in selecting distributed infrastructures -- both object-oriented and non-object-oriented -- we can confirm that almost every CIO has marching orders to implement "distributed objects everywhere." This begs the question: "Why aren't they?" The reason is, selecting a distributed object infrastructure is hard. It requires a strong understanding of a corporation's long- and short-term objectives, as well as excellent abilities to navigate through endless and sometimes conflicting vendor claims. You must also have a strong understanding of product categories, and of everything from application development to the intricacies of your company's business.

Some people define distributed objects as programming languages like Smalltalk, C++ and Java. Others equate distributed objects to object-oriented application development tools like Forté from Forté Software Inc., Oakland, Calif., and Dynasty from Dynasty Technologies Inc., Naperville, Ill. Some observers believe that selecting an object-oriented or non-object-oriented enterprise distributed infrastructure is a serious business that is separate from the choice of programming languages. This group assumes that development tools and distributed object-oriented middleware are as separate as apples and oranges. Granted, these product categories are different, but one of the most important decisions confronting all sides is choosing a distributed object middleware system. In short, the question is whether to choose the Common Object Request Broker Architecture (Corba) from the Object Management Group (OMG), Framingham, Mass., or the Common Object Model (COM) from Microsoft as a corporate middleware standard.

This article focuses on helping those information systems (I/S) organizations that have committed to using an object infrastructure to build distributed, multitier applications and are in the process of deciding which distributed object model to endorse. There are several questions to ponder besides whether to endorse OLE/COM, Corba, or both. And if Corba is selected, the organization has far more to do than just select one of the third-party implementations such as Orbix from Iona Technologies, Marlborough, Mass., PowerBroker from Expersoft Corp., San Diego, ObjectBroker from Digital Equipment Corp., and distributed SOM (DSOM) from IBM. The I/S organization must define the requirements for its distributed object infrastructure, and then decide whether the selected model alone can serve as a foundation for its distributed computing infrastructure.

Object middleware: Corba

First, Corba is a set of specifications published by the OMG, an industry consortium charged with defining a set of interfaces for interoperable object-oriented software. The Corba model is different from the Cambridge, Mass.-based Open Group's Distributed Computing Environment (DCE), which has a common source code base available on multiple platforms. Corba has no common code base, as the OMG does no software development. Corba is a set of specifications that specify the way objects are defined, created, instantiated, invoked and activated, and the way they communicate with each other.

The major components of the Corba specification are an Object Request Broker (ORB), an Interface Definition Language (IDL), an Interface Repository, an Object Adapter and the Common Object Services Specification (COSS), or CorbaServices.

At the heart of Corba is the ORB, a service application that functions much like a network switch setting up communication links and routing information between endpoints. (See Fig. 1.) The ORB provides the mechanisms for objects to make requests and receive responses. The Corba specifications require that all communication happen via the ORB. It does not, however, define how an ORB is implemented, so there are as many different ORB implementations as there are products on the market. Also left unaddressed in the Corba 1 specifications is a communications process for ORBs from different vendors. The Corba 2 specification addresses the issue of interoperability between different ORBs by defining a lightweight wire protocol called Internet Inter-ORB Protocol (IIOP). So all Corba 2-compliant products promise binary interoperability if they support the IIOP protocol.

In sum, the Corba specification can support any client object's request to any server object through an ORB. In other words, the ORB is really a broker between clients and servers, whose responsibility is to locate the designated server object and instantiate an instance of the object, if necessary. In addition, the ORB manages the object references required when client applications invoke server operations. However, it is up to the application to decide the contents of an object reference and how the reference was acquired in the first place. In general, the Corba specifications are more focused on defining interfaces and behaviors than on implementation issues.

The Corba specifications define two processes for accessing objects -- through a static invocation interface (SII), or through a dynamic invocation interface (DII). SII implies that the client object must know the IDL interface of the server object at compile times. In other words, the "contract" between the clients and servers is known and fixed. Any changes to the server interface require a recompile.

One of the most important components of the Corba specification is the IDL, a high-level universal notation language capable of defining interfaces that have attributes, operations, user-defined types and exceptions. The IDL specifications are compiled in what are called client stubs and implementation skeletons. (See Fig. 1).

The IDL specifications say nothing about the actual implementation of an interface. All Corba products generate language specific bindings based on the IDL. So far, the list of Corba-compliant languages are C, C++ and Smalltalk. Meanwhile, work on Ada 95, Java and Cobol mappings are underway. Generally, IDL definitions are stored in a public interface repository that typically contains all of the definitions for the system.

The Corba SII is similar to a remote procedure call (RPC) interface. As in RPC, the IDL interface is input to an IDL compiler. The code generated on the client side from the IDL specification is responsible for marshaling the operation parameters, such as to perform coding and decoding, into communication formats suitable for transmission across a network. The code generated from the server side is responsible for providing the callback to a server-supplied business function. Although SII is not exactly the dream of flexibility, it is very simple and therefore is used much more than the DII.

In sum, an IDL interface defines attributes and operations. Interfaces represent contracts between client and server applications. Each operation has a name, parameters and a result.

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, there is no blocking and the client application is free to do whatever it likes. However, it has to periodically poll the ORB for a response from the object, which executes the request. There is no provision in Corba for features such as automatic event notification or asynchronous queue-based communication that some message-oriented middleware (MOM) products provide.

Object middleware: COM

From a basic functional perspective the Common Object Model (COM) is similar to Corba. (See Fig. 2.) It supports the creation and destruction of objects, and can define, dispatch, traverse and interrogate object interfaces. Further, there are unique object references called GUIDs in COM, and an IDL. Similar to the Corba IDL, Microsoft's IDL supports additional functionality and the COM interface, which is defined as a family of logically related operations or methods.

There are also substantial differences between these two models. Some of the differences are as follows: COM is a binary model, which supports binary compatibility among different objects that have been written to the Microsoft "binary rules" specification; COM does not support multiple implementation inheritance -- only interface definitions, not implementations, can be inherited; COM is language-independent; COM is a product, not a set of specifications; COM is bundled with Windows NT, there is no extra cost for acquiring and using the technology; COM has less emphasis on an IDL; and COM has support for late binding that frees the client application developer from dealing with IDL compilers and binding stubs into the application, since all requests to server objects are built dynamically.

The late binding in OLE/COM is possible due to its OLE Automation component. OLE Automation allows a client to dynamically build a request comprised of the name of a method, and the types and values of associated parameters, as well as to issue it against a remote object. As with Corba, the client can obtain all required information about object interfaces from a repository. OLE/COM-compliant objects can also be interrogated about the interfaces they support at runtime.

Unique OLE/COM features

As indicated, there are three features of COM that do not appear related at first view: binary compatibility of COM-compliant objects; support for multiple interfaces per class; and the restrictions on inheritance. Together, however, they fulfill some very important requirements for the emergence of a successful, large commercial component market.

Off-the-shelf components will not become a viable alternative to in-house development until they become independent of the customer's target programming language environment. A software vendor's chosen component development platform must not be required to be the same as the user platform. Without binary compatibility, the component vendor would have to distribute source code to the market, an impractical strategy.

Once a corporate I/S shop buys and embeds a component in an application, it would be disastrous if the next version of the software included changes in the inheritance graph, such as a modified function or method in one of the component's super classes. This change would break the application, because it must rely on the exact functionality of all super classes it inherits. In addition, users would find it extremely difficult to use multiple component frameworks simultaneously, or, in other words, to use multiple inheritance.

The solution lies in the restricted support for inheritance in COM and the multiple interface concept. Whenever changes are made to a component, the original interface cannot be changed -- it remains stable forever. The component developer has to create a new interface that provides the new functionality. Support for multiple interfaces allows that.

Corba proponents are quick to point out that COM's restricted support for inheritance means it cannot be called object-oriented. Rather, they say, COM is object-based. However, we believe that corporate users are more interested in installing a distributed object infrastructure that offers solutions to their problems and can provide an ability to quickly develop applications that give them a competitive advantage. A debate over whether an infrastructure is object-oriented or object-based is of little relevance to most corporate developers.

Though the paragraphs above represent just a partial list of the similarities and differences of COM and Corba, we believe that both technologies are important. An important issue is the question of whether these technologies can work together.

But before we answer this question, we will look at some other, but no less important issues, such as the questions of whether distributed object middleware is for everyone and whether it is mature enough to build a solid enterprise foundation for scalable, multitier, distributed applications.

When object middleware is the right choice

Companies in a variety of industries, with projects ranging from manufacturing shop floor control to applications in the financial world, have chosen to use object middleware. The requirements of these organizations and the successes of their middleware efforts vary widely. Nonetheless, it is safe to say that those corporations that have successfully implemented distributed object middleware tend to be deep pocketed and have leading edge I/S personnel. Generally, these companies are in extremely competitive industries such as investment banking, telecommunication and aerospace. Some of these industries require a quick response to quick changing government and nongovernment regulations, and most must also quickly introduce new products and business initiatives to stay ahead of the competition, which means that new business applications must be in place quickly. In turn, the applications must use either prefabricated components, or leverage existing applications where a substantial amount of code can be inherited and reused.

Below we list a few scenarios to consider when embarking on an expensive and risky journey to distributed object middleware.

The perfect fit

An organization that has already developed object skills, particularly in C++ or Smalltalk and OO analysis, design and development toolsets, is in the best position to take advantage of object middleware. The company can utilize object-oriented languages and tools and avoid the procedural languages that are supported by Corba and OLE/COM products, but are cumbersome to use and cannot fully exploit the capabilities of object orientation.

In addition, with experience in developing and deploying multitiered distributed systems, I/S developers are already familiar with the concept of IDL, although they will have to embrace a new, object-oriented flavor of IDL. The target project for such an organization should be implemented using object-oriented concepts like class structures. The unit should design a multitier architecture and document the interfaces in IDL, or via a repository-based tool.

A good fit

We have seen a number of sites using object middleware primarily as a systems integration tool. Many of today's applications look like spaghetti -- a messy web of point-to-point connections between applications without standardized interfaces. Object middleware can bring order to this chaos. Each existing application, or functional part of it, is encapsulated or "wrapped," creating a standard interface structure that can be plugged into the middleware, making the application available to any client on the network.

This approach is usually part of a migration strategy. Once applications are turned into this kind of component, they can be enhanced with new user interfaces, they can be reused, and eventually they can be replaced with little disruption to regular operations.

A poor choice

Most business functions are still developed using procedural languages like Cobol. There is no need to use object middleware on a non-object-oriented application. There are several alternative middleware technologies, such as an RPC-based system like DCE or MOM. MOM fits best when different communication styles and interactions between application components are required. For example, if the client application should continue to interact with the end user instead of having to wait for a response from a request to a server. To date, most Corba products do not provide adequate support for a variety of communication mechanisms.

Building a solid foundation

The basic middleware services in Corba and OLE/COM products may not be sufficient to build an infrastructure to support a multitude of new applications, particularly enterprise applications that must scale and support a substantial number of clients and servers across large, geographically dispersed networks. This is especially true for mission-critical applications requiring 7x24x365 availability and split-second response time.

Some essential middleware services such as naming, events, life cycle, persistence, security and transactions have been defined by the OMG, and have been collectively called CorbaServices. However, the OMG has only defined the interfaces of these services. There is no reference implementation and vendor support for these services varies widely.

There are also a number of services that are not part of the OMG standards or the current OLE/COM product. These include load balancing, fault tolerance, message queuing, publish and subscribe mechanisms, high scalability, and CICS and IMS support.

Many Corba vendors understand that their products must support at least some of the CorbaServices and other features to compete with some of the more mature non-object-oriented middleware products.

Security and naming

Corporations also need to consider security and naming services when selecting distributed object middleware.

The security capabilities of Corba products in the market today vary widely. Some, like ObjectBroker from Digital, are integrated with DCE security services. ObjectBroker supports the Generic Security Services API (GSSAPI), supporting both DCE-based security (Kerberos) and third-party authentication packages such as CyberSafe.

A distributed middleware infrastructure also needs a comprehensive naming service. In fact, a naming service is essential for any middleware solution. A naming service is required for location transparency, an important requirement, especially with the increasing development of more large-scale, truly distributed applications.

The OMG has adopted a standard naming service interface as part of CorbaServices. However, most Corba-based products are implemented using proprietary naming, varying widely among different Corba vendors. For example, none have implemented name space replication or federated naming. SuiteDome from Suite Software, Anaheim, Calif., is probably the only object middleware product to implement replicated and federated name space. However, SuiteDome is not yet Corba-compliant.

Good implementations of naming services are found in other middleware offerings, such as DCE and the Pipes Platform, from PeerLogic, San Francisco, however. An attempt by vendors of other middleware products to leverage existing naming services could be beneficial. For example, OLE/COM has its own naming service implementation, and Microsoft uses "monikers" as intelligent names. And a fully distributed naming service will be incorporated in Windows NT 5.0.

Transactions

Certain applications require transactional integrity. For example, many financial applications can require that account information be either updated in two or more databases, or none at all in a single transaction. If a transaction is interrupted, partial updates have to be rolled back.

There are no out-of-box solutions in this area today. Object middleware is just now being extended with transactional capabilities. The OMG has defined an Object Transactional Service (OTS), and it is up to Corba vendors to implement it. Unfortunately, many Corba vendors do not yet possess the skills required to develop OTS. Therefore, we expect joint development efforts and partnerships to provide the solution. For example, Iona has agreed to ship OTS as part of its agreement with Transarc Corp., Pittsburgh.

Implementation of OTS in Corba products can allow business applications developed with Corba to update multiple heterogeneous databases in a single transaction. However, OTS does not automatically provide Corba products with features generally offered by distributed transaction processing (TP) monitors. For example, failover, load balancing and sophisticated naming services traditionally found in TP monitors will not be automatically supported by object middleware. We believe that only one product, SuiteDome, provides many of these services, along with bullet-proof security.

On the OLE/COM side, Microsoft will soon be unveiling a new technology called Viper, which is said to add transaction capabilities to its DCOM distributed object middleware. Viper is expected to allow programmers to build server applications as components in languages like Visual Basic. Such components will be less complex than the TP monitor systems and therefore can be sold at a low price. For example, Viper will not require programmers to deal with threads.

MOM integration

As suggested, there are several middleware services and features that are essential for a distributed infrastructure. Some are specified by the OMG and some are currently outside the OMG specs.

There is no Corba product on the market that supports all of the essential middleware services, including load balancing, fault tolerance, message queuing with guaranteed message delivery, and support for MVS subsystems like CICS and IMS. However, many of these are available in other types of middleware. For example, Iona’s Orbix supports fault tolerance through integration with ISIS, transaction processing software from the ISIS unit of Stratus Computer Inc., Marlborough, Mass.

High scalability is a strong feature of the MOM products, many of which also have built-in features such as session concentration for applications that do not communicate directly. From an application point of view, only queue managers communicate with each other, allowing multiple application processes on one machine using a single queue manager to communicate with application processes on another machine, which also is using the services of a single queue manager. This feature, called session concentration, is a key to scalability.

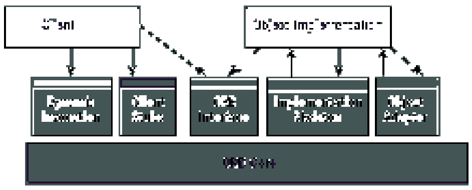
Some MOM products also offer message queuing with guaranteed message delivery, events, publish/ subscribe and load balancing. For example, OLE/ COM does not support

MVS, and Corba offers only very limited support, while most mission-critical applications are still MVS-based. The missing link to these environments, particularly CICS and IMS, can be established by integrating object middleware with message-oriented middleware. A few leading edge end users, usually with help from consultants and systems integrators, have successfully implemented a combined MOM-object middleware system. Some firms have implemented combinations where an object on a Unix system acts as a middle tier connecting to mainframe applications via message queuing middleware. (See Fig. 3.)

System integration and custom interfaces combining best-of-breed middleware is a viable strategy while waiting for fully integrated products delivering all key services. Meanwhile, Iona Technologies is integrating Orbix with IBM's MQSeries, allowing Orbix objects to communicate asynchronously as MQSeries provides services such as guaranteed delivery of requests and replies across any type of network, including intermittent networks. 1

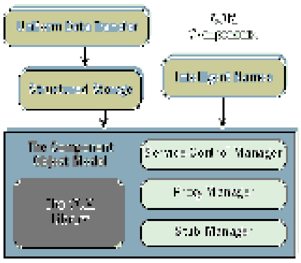
Next: Platform requirements, OLE/COM and Corba integration, and standards.

Corba Components



Source: International Systems Group Inc.Fig. 1 The major components of the Corba specification are an Object Request Broker, an Interface Definition Language, an Interface Repository, an Object Adapter and the Common Object Services Specification.

COM Building Blocks



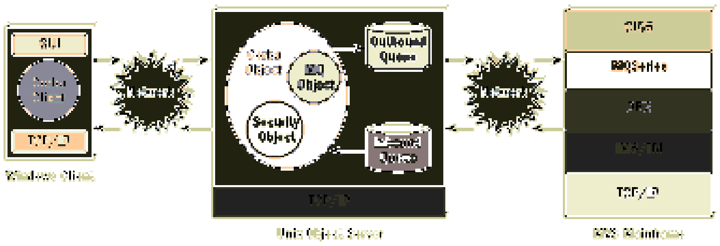
Source: International Systems Group Inc.Fig. 2
COM supports the creation and destruction of objects, and can define, dispatch, traverse and interrogate object interfaces.

Once a corporate I/S shop buys and embeds a component in an application, it woud be

disastrous if the next version included changes.An organization that has already developed
object skills is in the best position to take ad-vantage of object middleware.

Objects & MOM Together

Sourc
Interna
Systen
Group
Inc.Fi
3
Some
firms



have implemented successful MOM-object middleware system combinations where an object on a Unix system acts as a middle tier connecting to mainframe applications via message queuing middleware.