Articles & White Papers

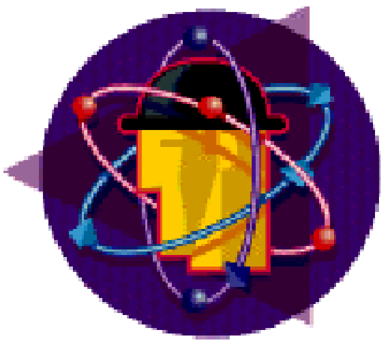
Object-Oriented and Component-Based Development

Strong showing of DCOM and viability of Corba leave corporate developers with the daunting task of integrating two technologies

DCOM, ActiveX and Corba Must Live Together

by max dolgicer and peter fischer

illustration by Court Patton



Companies that are committed to the development of applications using distributed objects or assembly of applications from discreet, self-contained components have the potential to gain substantially over their competitors. The world of object technology is hot, and in the appealing class of technology known as middleware, object middleware is the fastest growing category.

For corporate developers and independent software suppliers, object middleware has tremendous appeal due to its ability to provide highly abstracted programming interfaces. This contrasts it to other categories of middleware, such as message-oriented middleware and some distributed TP monitors, which furnish developers with relatively lower level APIs. Object middleware offers developers a standard interface in the form of the Interface Definition Language (IDL), which "speaks" about business objects, methods and interfaces. It further contrasts object middleware to other categories of middleware in that no communication code has to be developed; when IDL compiles, the communication code required for clients to communicate with servers is generated.

Organizations committed to developing applications using object middleware must decide whether the Framingham, Mass.-based Object Management Group's Common Object Request Broker Architecture- (Corba) compliant products or Redmond, Wash.-based Microsoft Corp.'s DCOM are the right object-oriented (OO) technologies for their business applications.

Despite substantial inroads made by the NT operating system -- the availability of Microsoft's OLE/DCOM-enabled tools, and their very attractive "price" (DCOM is bundled with the NT 4.0 operating system) -- the wide acceptance of OLE/DCOM as the "right" object technology is being hindered. This is due to its present lackluster support for non-Windows operating systems as well as its lack of some robust middleware services. Although, OLE/COM will be generally available outside of Microsoft operating systems (thanks to the "German engineering" being performed by Software AG, Reston, Va.) on leading Unix platforms, many remain skeptical that OLE/DCOM will successfully penetrate the non-NT market. Given the plethora of development tools which are tightly integrated with OLE/COM on Microsoft operating systems only, it is hard to arge such skepticism is unreasonable at this time.

On the other hand, despite the fact that Corba-compliant products such as Orbix from Iona Technologies Inc., Dublin, Ireland, ObjectBroker, formerly from Digital Equipment Corp., Maynard, Mass., and recently sold to BEA Systems Inc., Sunnyvale, Calif., and others are available on a wide range of operating systems, Microsoft's OLE/DCOM is hard to ignore. Assuming that the statement "the world will never be all NT" is correct, Corba is viable. The bad news is corporations are left with the daunting task of integrating both of these important technologies together.

This feature article examines some of the solutions available for integrating OLE/DCOM with Corba-based applications. It describes the pros and cons of each solution, and highlights the real world problems to be addressed.

Terminology update

In the technology industry, the one constant is change, and the terminology surrounding OO middleware is not an exception. In positioning its DCOM infrastructure for the Internet and Intranet, Microsoft added some new terms and changed the meaning of some existing terms. The Component Object Model (COM) is still used to describe the base model used for building components , and the standard that governs the interaction of these components at runtime. The term DCOM is being used to describe the distributed form of COM. In this article we interchangeably use the terms DCOM and COM. ActiveX is the packaging technology for controls and was formerly called OCX, which is to supercede Visual Basic controls, VBXs. The original term OLE is not an acronym and does not mean object linking and embedding any longer, and it currently refers to a collection of technologies. Finally, the programmability interface once known as OLE Automation, has been shortened to just Automation. Thank you Microsoft for all the changes!

DCOM and Corba: Two paths

Today, Corba is seen as the object-oriented middleware of choice when enterprise systems need to support distributed applications that run on a variety of operating systems. Because of this, Corba tends to be used for the corporate abstraction of business objects and processes. In addition, because of the connectivity of Corba products across a mix of platforms, Corba-compliant products are often used as the communications middleware. Corba can supply a large-scale object-oriented framework to support enterprise mission-critical systems. It achieves this by providing the business object communication infrastructure through the object request broker (ORB), and value-added object services through what is now called CorbaServices, formerly Common Object Services (COSS). CorbaServices, as defined by OMG, are essential for the enterprise development and deployment of distributed applications. These include services such as naming, events, transactions, persistency, security and many others. Some Corba-compliant ORB products today support a selection of these defined services.

Corba has its roots in the network and is built from years of experience with TCP/IP. It has strong support for IDL and exception handling, strong typing and a language-based approach for creating objects.

On the other hand, DCOM has its origins in the desktop and builds heavily from the Windows programming model. This includes DLLs, its binary model and its underlying RPC mechanism for distributed capabilities. In fact, when COM first appeared on the scene it was intimately intertwined with OLE, which at that time stood for Object Linking and Embedding. Hence, its roots really are derived from a compound document technology. Only after OLE 2 came out was it recognized that OLE could be used for more than compound documents, hence the split into OLE and DCOM. At times, the compound document heritage of COM is evident.

The interface philosophy and the way clients access Corba servers differ significantly between DCOM and Corba. A Corba object has one interface that is comprised of all the interfaces of the sub-classes. In contrast, the DCOM object has multiple interfaces. A client of a Corba object has full access to the whole interface of the object as in a traditional language-based object, such as C++ which means that given a reference to an object, client code can access every publicly available method and property that the object exposes. On the other hand, all access to a DCOM object is through interface pointers, which only let a client access a particular set of functionality at a time.

In addition, the foundation of DCOM is built upon a robust, binary interoperability standard (real plug-and-play) between COM components. Meanwhile, the Corba interoperability is limited to the recent Internet Inter-Orb Protocol specification (as part of General Inter-ORB Protocol), which spells out how Corba objects developed with different Corba products can interoperate with each other. These lead to the fact that DCOM is language-independent, while Corba provides language specific bindings. The issue here is that if the ORB does not support bindings to a specific language, the user will not be able to develop object servers and object clients using that language. It should be mentioned, however, that many Corba products support multiple languages and support for additional languages is not difficult to provide. It would require the generation of additional language stubs from the IDL definition -- something manageable for Corba vendors who are familiar with the intricacies of generating client and server stubs for specific languages.

DCOM uses two mechanisms to expose the functionality of its objects. The first is the custom interface, which is a set of semantically-related methods that carry out a specific set of functionality and therefore, related methods. Typically, 3GLs such as C++ access DCOM objects through custom interfaces.

The second method is Microsoft's Automation mechanism. This provides for the ability of a DCOM object to expose its behavior through a generic interface called IDispatch. This allows any object to expose its functionality as a series of methods and properties to a client object that "speaks" IDispatch. The Microsoft Automation is supported natively by many application development environments that have deep market penetration. These development environments include, but are not limited to, Visual Basic from Microsoft, PowerBuilder formally from Powersoft Corp., Concord, Mass., now Sybase Inc., Emeryville, Calif., and Delphi from Borland International Inc., Scotts Valley, Calif. Wrapping COM objects as Automation objects provides the greatest potential for reuse without locking the users of these objects into a particular development environment. For example, a PowerBuilder programmer would want to have access to functionality represented by Automation from PowerScript. Some 4GL-based tools such as Visual Basic and Delphi will support the creation of ActiveX controls which also "speak" Automation.

A recent development in DCOM has been the introduction of dual interfaces. This is a sort of one-stop-shopping for correct and natural access to an object's methods and properties. C++ code can naturally access the objects through its built-in vtable mechanism (a vtable is an in-memory table which contains pointers to all of methods of the interface of the C++ class), while Visual Basic code can access it through the IDispatch interface, which is natively supported by the Visual Basic environment. The dual interface is derived from the IDispatch interface, and in most cases, the dual interface can use the same code used by the IDispatch interface, that code can delegate its implementation to that of the IDispatch code. The advantage is the business object developer can only write one codebase.

In sum, DCOM with dual interfaces provides "many natural paths to the object." With Corba alone, the choices of object clients accessing object servers are more restricted. They are limited to what traditional language-based object-oriented languages, such as C++, Smalltalk and Java, provide. This is exactly one of the areas where the OLE/COM-Corba interoperability makes tremendous sense. As will be discussed, Corba server objects can easily be accessed from any OLE/COM-enabled technology, Automation being a good example. This immediately makes Corba objects accessible by a plethora of development and personal productivity tools.

Why the debate?

Most CIOs understand the importance and the need for distributed objects. Many view distributed objects as a vehicle for migrating legacy systems into the new distributed object enterprise. The factors driving the decision to go DCOM or Corba are blurred by the introduction of new technologies and the capabilities attributed to each.

It used to be so simple. COM, because of its desktop heritage, was limited to desktop implementations, while Corba, because of its wide support for many platforms and built-in distributed capabilities, was viewed as the enterprise solution. COM provided no support for distributed capabilities. By default, if you needed distributed object functionality, Corba was the only game in town. However, to be an enterprise-class middleware you need to provide support for enterprise-class distributed services. Being just distributed is not enough, and most Corba products lacked support for such services. As a result, a number of cutting-edge end-user organizations had no choice but to develop such services in-house, thus supplementing the basic Corba with additional functionality.

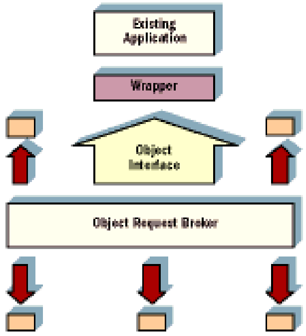
Currently, such black and white decisions are being blurred into many shades of gray. Corba products continue to mature in their support for naming, transactions, security, event and other services defined by the OMG. At the same time, the enterprise components of the DCOM puzzle fall into place with delivery of Microsoft's Transaction Server, messaging (project code name Falcon), and the forthcoming next release of the NT operating system. However, the concerns about heterogeneous platform support are not melting away even as we come closer to having Unix versions of DCOM. The questions of parity of COM's future functionality between NT and other platforms, and the integration of COM with development tools still persists. It used to be simple. By default, the choice of the server side object-oriented middleware was Corba: this choice was not obvious, especially when the server operating system is NT. DCOM is coming on strong and cannot be ignored. On the client side, the situation is quite predictable. Since Microsoft is "king of the desktop" in terms of the operating system and the end-user personal productivity tools, many businesses are standardizing based on the "Microsoft everything desktop." As a result, DCOM is the middleware of choice when the majority of the integration of functionality is at the desktop, in the hands of empowered users and business solution developers. These developers are comfortable with the ease of use and the responsiveness desktop development and personal productivity tools offer.

One other thing about DCOM is its price -- it is free. Being an integral component of the operating system offers distinct advantages, as well as a consistent interoperability model between object services and operating system services. The bottom line is that when it comes to the middleware of choice for the client side, the debate is almost moot. However, for the server side, the DCOM versus Corba debate is being brought to life. In fact, the quick product cycles that leading Corba products are experiencing, combined with the fact that Microsoft is on the fast track to releasing serious middleware products (such as Microsoft Transaction Server and Messaging), will make for an interesting future. At least in theory, I/S managers are faced with the "DCOM versus Corba" object middleware dilemma in which the right choice is not obvious.

The reality -- no choice

In practice, many I/S managers cannot choose between the two alternatives. They need to work with both object models in order to deliver technical solutions to business problems. Corba is a good choice for enterprise solutions because of its broad platform support, while DCOM is viewed as the solution for providing the integration point for applications on the desktop.

Corba and legacy apps



When a legacy application is "wrapped" as a Corba object, it enters the realm of distributed objects.

Fig. 1

Source: International Systems Group Inc.

In large corporate environments, a bulk of the business processing is done on high-end operating systems such as MVS and Unix. Many of these systems, often referred to as legacy systems that cannot be extended without a major rewrite, run line of business systems today and contribute to the corporate bottom line. A major trend in corporate development has been to use Corba-based products to "wrapper" these legacy systems, making them look and behave like objects, and enabling them to be integrated into the new distributed computing enterprise. (See **Fig. 1**.)

New development efforts are focusing on providing this functionality to the desktop. This is because most of today's easy-to-use productivity tools reside on the Windows platform, and corporate developers are using them to deliver business solutions. What is needed is a way to bring the two together and "push" the functionality of the enterprise out to the business solution providers, who need to access the functionality of these Corba-based objects from the desktop. Combining these two models at the desktop will enable the use of both models to support the business of the enterprise, and provide an overall end-to-end distributed object infrastructure.

As a consulting company that specializes in delivering enterprise-class distributed object systems, International Systems Group Inc. finds the problem is not Corba versus COM, but a business value proposition. It is an integration challenge to extract the power and capabilities of each model to solve the business problems at hand. Arguments of technical merits forwarded by purists in the respective camps, fall before the need to solve business problems.

Critical advantages and disadvantages		
Approach	**Advantages**	**Disadvantages**
Roll Your Own	*Provides the right solution -- always customizable *No vendor lock-in *You own the solution	*Labor intensive *Maintenance intensive *Always playing "catch-up" with technology
Vendor Proprietary	*Quick path to "object wrappering" and exposing functionality as Automation and/or language-based objects servers	*Vendor lock-in
Object Bridge	*Transparent *Vendors neutral *Provides many to many bi-directional communication *Robust exception handling	*Limited product supports *Cost *Complexity
OMG Compliant	*Provides future migration path when products become available	*Products are not on the market yet

Table 1
Source: International Systems Group Inc.

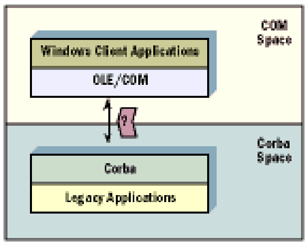
There is no one-size-fits-all solution, and probably never will be. The solution is to use the correct combination of the technologies to deliver the business solution. In order to deliver on the promise of true distributed object computing, an enterprise-class infrastructure must combine the use of important and necessary technologies.

Four integration alternatives

Enterprise developers currently have four alternatives for delivering an integrated mixed-model solution, going from DCOM to Corba and vice-versa. (See **Table 1**.) These range from an in-house RYO (roll-your-own) or DIY (do-it-yourself) solution, to purchasing components called object- bridges that map transparently between the two models. The integration of the two models really is an exercise in mapping interfaces and parameters between the two.

The basic problem is converting from "COMSpeak" to "CorbaSpeak" and vice-versa. (See **Fig. 2**.) Logically, a component must sit in the middle to translate interfaces and parameters between the two models.

The basic mapping problem



Integrating COM and Corba models requires replacing the question mark.

Fig. 2

Source: International Systems Group Inc.

In selecting the correct alternative, several important considerations need to be kept in mind. One consideration is the current skillset of your development staff which cleary represents the greatest amout of risk if your development staff is not technically advanced and not familiar with Corba, COM and object-oriented languages such C++ or Java. Time frame for production is another consideration, and the list of considerations can go on.

However, corporate developers should consider the way that COM and Corba objects are capable of communicating with each other. Owing to Microsoft’s desktop dominance, most implementations so far have been concerned with COM components calling Corba objects. This is one way, or unidirectional communication, which is supported by a number of the Corba products currently on the market. Relatively few products support bi-directional communication whereby the Corba object can communicate back to the COM component.

Roll your own

Companies that cannot wait for OMG-compliant products to hit the market, and want to avoid the use of proprietary solutions for fear of vendor lock-in, have chosen a roll-your-own solution (RYO). This enables them to select "best-of-breed" products off-the-shelf and build a distributed object infrastructure over them. They are then able to slot in new products as they become available on an as-needed basis.

Although this approach tends to involve more work effort, it also provides a valuable learning experience for the development staff in seeing how the technologies play out. The experience gained from integrating DCOM and Corba together provides an invaluable asset for the company and its personnel as the move to distributed objects hits full force. By using straight Corba IDL to wrap their objects, there is no vendor lock-in. (See "Brokerage firm developers use Corba IDL to ’wrapper’ useful legacy code and push it out to Microsoft desktops.")

In addition, these solutions tend to be more customizable than solutions created from proprietary products. There is more control over how the objects are built, and how the functionality is partitioned, in addition to how the COM interfaces are named and generated. The COM object implementation is able to completely encapsulate any ORB-specific calls, protecting the client applications from ORB implementation details and the investment in business solution code.

A few Corba-compliant products offer an interface that allows client objects to talk to their Corba-based objects via either an Automation IDispatch-based interface and/or custom interface. For example, Iona Technologies includes a product called OrbixOLE together with their industrial-strength Corba-compliant Orbix product. This product generates all the code necessary to marshal parameters between COM and Corba. Client objects call into the server object using natural Automation calls.

OrbixOLE is quite a complete product with the look and feel of native Microsoft product. It takes a lot of the grunt work out of the development process, and its substantial appeal to developers derives from its ability to generate all the necessary files to support COM and Automation invocation (IDispatch interface, for example) without having to write much code. The IDL compiler is instrumented to generate COM stubs from Corba IDL and it also creates Corba stubs, which "talk" to COM interfaces. It is also one of the few products currently available that provides support for both IDispatch and custom interfaces. As a result, millions of Visual Basic, PowerBuilder, Delphi, Internet Explorer as well as C++ developers are provided with an easy way of accessing Orbix objects. Lastly, OrbixOLE provides bi-directional communication support where either COM or Automation objects can call Corba objects, or vice versa.

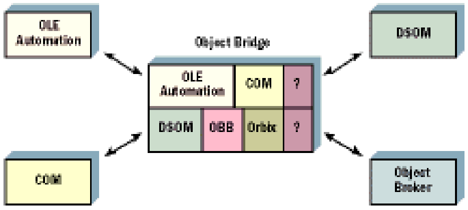
Additional vendors such as BEA Systems' ObjectBroker, Visigenic's VisiBroker and others provide proprietary solutions for integrated OLE/COM withCorba.

An advantage of the vendors' proprietary integration approach is that it provides an integration solution between OLE/COM with specific Corba products today. The drawbacks are lock-in at the Corba product level and the fact that propriety, vendor-specific code is being generated. For example, the Automation objects generated by the Orbix IDL compiler require a call to specific Orbix methods for binding to object servers. Thus, it is possible that any client code that calls these Orbix Automation proxies will have to be rewritten if the underlying Corba product is changed.

Object bridging

Conceptually, an object bridge is a piece of software that acts as a bridge which converts a call from one object model into a call to another object model. Borrowing a term from the database world, it can be considered an "object-gateway." One example of such a product is Object Bridge, developed by VisualEdge Software, St. Laurent, Quebec. Object Bridge is based upon Visual Edge's Object Systems Adapter (OSA) specification. OSA specifies an engine that is capable of converting requests from one object model, into another using what is called object adapters.

Visual Edge's Object Bridge



Object Bridge is a sophisticated engine capable of performing the mapping between different object systems.

Fig. 3

Source: International Systems Group Inc.

To achieve interoperability, each object product needs to implement an object adapter based on Visual Edge's specification. Visual Edge has already implemented robust adapters for Microsoft's model, including both the Automation and COM. As a result, by implementing the object adapter specification for its product, a Corba vendor will immediately be able to communicate with each of the object models (including Automation and COM) that have adapters already in-place plugged into ObjectBridge. For example, BEA Systems' ObjectBroker will be capable of communicating to any Automation or COM objects through use Object Bridge. (See **Fig. 3**.) This approach has a number of advantages.

First, the communication between different object models and products is bi-directional. Second, many to many interoperability is provided since each product plugged into the bridge gets immediate access to the object model and products which already have the adapters in place. The disadvantages of such an approach are quite obvious: the Visual Edge solution is proprietary and it is too early to say whether or not the initiative will catch on. Despite that, the specification is solid and the company shines with confidence. The ultimate success of the approach is in the hands of the vendors themselves. As all know, good technology is not necessarily always a winner. In addition, the implementation and development of such an adapter is complex; Object Bridge specifications are quite comprehensive. As of this writing, only a few Corba products have invested the engineering resources needed to develop such an adapters.

The Object Bridge alternative is probably a good choice for organizations that cannot wait for OMG specification-compliant products to hit the market. Obviously, if the Corba vendor of your choice is serious about supporting this approach, integration between its product and COM becomes straightforward.

The OMG has been fully aware that DCOM has made significant inroads as a distributed object model and has taken steps to see that the two can interoperate. A first step towards this is the OMG specification, which is called the COM-Corba Internetworking. This specification is concerned with support for bi-directional communications between Corba objects and COM objects. The goal is that objects from one model should be able to be viewed as if they existed in the other object model. The specification refers to this mapping as "views."

There are four common scenarios that are encountered for interoperability: exposing COM objects as Corba objects, exposing Automation objects as Corba objects, exposing Corba objects as COM objects and exposing Corba objects as Automation objects.

In fact, BEA Systems with its newly acquired ObjectBroker, had been developing a value-added product called the Desktop Connection. This product will provide second-generation connectivity between COM and Corba objects that will extend the capabilities present in today's ObjectBroker product. However, as of this writing this product is not shipping yet. When it ships, it will provide transparent bi-directional support between Automation and COM objects with ObjectBroker objects. It will allow out-of-the-box Win32-based OLE applications to access unmodified ObjectBroker objects via Automation, ActiveX, drag-and-drop and linking. This will seamlessly integrate ObjectBroker and COM objects into OLE Document Containers objects for visual rendering, in-place activation, cut-and-paste and drag-and-drop without additional coding.

The bottom line is that enterprises interested in integrating OLE/COM and Corba, according to OMG specifications, will have to wait until products compliant with the specification hit the market.

Conclusion

Implementing distributed object technology for enterprise-class systems presents many challenges and alternatives. The correct choice is never at either end of the spectrum, but falls somewhere in the continuum that lies in between. Where it falls depends on both the business and technical problems that need to be solved.

Development managers need to do their homework carefully, and plan an evaluation before jumping into the sea of distributed objects. The skills required to master development of applications using OLE/COM or Corba are essential and have to be imported into the organization as required. The technology is changing all the time and very rapidly.

Max Dolgicer is a director at International Systems Group Inc. (isg@isg-inc.com), a New York City-based consulting and research firm that specializes in the design, development and implementation of complex distributed applications using middleware. Mr. Dolgicer can be reached by E-mail at mdolgicer@isg-inc.com (corporate) or mdolgicer@aol.com (personal).

Peter Fischer is a senior consultant at International Systems Group Inc. With over 10 years experience in developing distributed systems, he is a recognized authority on implementing distributed object systems using DCOM and Corba technologies. Mr. Fischer can be reached at pfischer@isg-inc.com.

Brokerage firm developers use Corba IDL to "wrapper" useful legacy code and push it out to Microsoft desktops

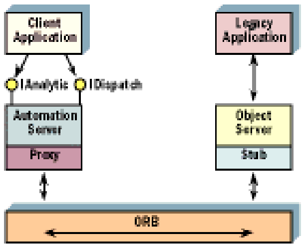
Brokerage firms are concerned with the inherent risk associated with holding a set of securities. Due to market volatility, the value of their holdings can dramatically change over a short period of time and thus quickly alter the fortunes of the firm. To protect themselves from risk, brokerage firms use a series of analysis and calculation routines that enable both brokers and traders of the firm to evaluate their portfolios. Such Risk Management Analysis is done on both a daily basis with the Profit and Loss (the so-called P & L), and over time using time curve analysis routines.

One major Wall Street firm faced the challenge of empowering traders and brokers with the ability to use Microsoft Excel to do their analyses. This firm has a huge investment in Microsoft technology on the desktop and the users were enamored of the functionality that Excel provided. The struggle facing the firm’s enterprise developers was how to make the functionality that resided in different server platforms available to the Excel developers.

This business problem was viewed as an opportunity for introducing object technology to the company. This firm took up this challenge by creating an architecture that provided the risk management functionality in the form of a series of multitier business objects, thereby "objectizing" the enterprise.

Because it was an early attempt by the firm to move to distributed object technology, the technical team sought to avoid vendor lock-in with proprietary products and to gain an in-depth knowledge of how to develop distributed object applications. A goal was to build both a base of knowledge and a technical infrastructure upon which to build future systems and to provide extensibility to the system being developed.

The Runtime Architecture



Client applications call the legacy application through an IDispatch or custom interface which is then routed to a Corba server which interfaces with the legacy application.

Fig. 4

Source: International Systems Group Inc.

The existing analytic routines running on Unix midrange boxes were wrapped as objects using Corba. In doing this, the legacy applications could be leveraged, and they still provided functionality for the existing business systems. On the client side, these objects were wrapped as COM objects using both Automation and custom interfaces. This allowed the functionality provided by the analytic to be pushed to the Microsoft desktop and made these business objects accessible from a multitude of languages and productivity tools, such as Excel and Word. (See **Fig. 4**.)

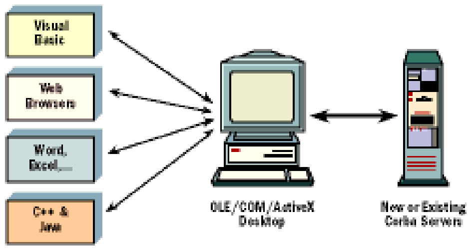
The development effort was split into two parts -- client development and server development. The legacy applications were to be wrappered as object servers to be plugged into the new object-oriented infrastructure being put into place.

First object-oriented analysis and design was done to create the object model from the legacy applications. From the model, the developers generated IDL. Running these IDL files through the IDL compiler that came with the ORB yielded proxy and stub files that were needed to create the client and server objects, respectively. The files provided the communication abstraction for the specific ORB, allowing the developers to focus in on "filling in the blanks" with code.

The object servers on the Unix side were built using the stubs generated by the IDL compiler. These objects provided the interface between the legacy system and the clients by mapping calls from the new object interfaces to the legacy applications. The server stub methods were filled in with calls to the legacy applications. The calls returned data from the legacy applications to the client application.

The client side Automation server was built from the proxies generated from the IDL compiler. Using Visual C++ and the ClassWizard provided a rapid way to create the IDispatch interface and empty methods to implement the Automation methods. The bulk of the work was then simply implementing these methods by calling through to the appropriate object server method that performed the functions. In order to reference the remote object server, an instance of that server was created in the Automation server initialization code.

OLE/COM-Corba combination



Integration for OLE/COM-based desktop client applications with enterprise Corba servers present a powerful combination.

Fig. 5

Source: International Systems Group Inc.

The business solution code was written in Microsoft Excel using Visual Basic for Applications (VBA) to access and manipulate methods on the Automation server. These methods returned the data that was needed for the spreadsheet analysis that was carried out by the traders. These methods were packaged into Excel macrosheets and the charts and graphics were incorporated as worksheets. All the trader had to do was fill the appropriate data in the worksheet and hit the Start Analysis button.

The VBA code then performed the calculations by calling the methods on the Automation server, which called through to the object server on the Unix side, which in turn called into the legacy applications. After the results were returned to Excel, the charts and graphics were automatically run. This development laid the ground work for any technology on the client (not just Visual Basic) to be able to access new or existing Corba servers. (See **Fig. 5**.)

The big win for the development team was that this approach produced the right result. During a post-mortem examination, the team reached several conclusions. First, they learned that their chosen solution was labor-intensive, in that it required them to craft a significant amount of code in both the Automation and object servers.

The knowledge gained from the effort was time well spent. Members gained an understanding of ORBs, COM and how the technologies work, as well as an appreciation for the power of distributed object technology. The team authored a repeatable process to complete the project, as well as a baseline which could be extended with future functionality.

The team generated code that will lessen the implementation effort for future endeavors. The fact that the basic business functionality code was separated from the ORB-specific code, enabled the reuse of this codebase with other Corba-compliant ORBs. Thus the team avoided vendor lock-in, a significant concern going into this project.

The company's big win was the ability to access those server objects from a variety of client environments. Although this implementation used Excel as the client, more robust solutions using Visual Basic and Visual C++ are under development. Because those server objects were packaged as Automation objects with dual interfaces (i.e. support for custom, C++ interface), they were accessible from both of these languages.

-- Max Dolgicer and Peter Fischer