COM+: The Evolution of Component Services

Guy Eddon, Learning Tree International

The Windows platform has, in a few years, evolved from a personal computing OS to an enterprise solution. Key in this strategy is the set of component gluing technologies that Microsoft groups under the general term DNA. In this column, Guy Eddon describes the evolution of a key component of this approach, COM+, the most recent incarnation of the COM component model.

—Bertrand Meyer

COM+ encompasses two areas: a fundamental programming architecture for building software components (first defined by the original COM specification) and an integrated suite of component services replete with an associated runtime environment. For many developers, however, the fundamental COM model is insufficient.

In the typical corporation, developers build business components that operate as part of a larger application. Developers expend a great deal of effort even to build the simplest of components. They must also create a robust and secure framework for it and ensure that only clients with the proper authorization can perform certain privileged operations using the component. The problem with the bare-bones component model is that developers are left to implement an enormous amount of functionality that has little to do with the goals of the application itself.

The problem with the bare-bones component model is that developers are left to implement an enormous amount of functionality themselves.

To reduce the amount of work required to build a complex distributed application, Microsoft developed the Microsoft Transaction Server (MTS), the first Windows-based implementation of a runtime environment to provide component services. These component services operate within a general set of tools and technologies that, together with COM+, form what Microsoft calls the Distributed interNet Applications (DNA) architecture.

WINDOWS DNA

Because Windows DNA provides a comprehensive and integrated set of component services on the Windows platform, developers are free from the burden of building or assembling the required infrastructure for distributed applications. The goal of DNA's approach is to separate the business logic from a client-server system by moving it to a middle tier that runs on Windows 2000. The resulting three-tier architecture consists of a presentation layer, the business logic components, and the data services layer.

Presentation

The client side of a client-server system typically encompasses the functionality of both the user interface and the business logic that drives the system, leaving only the database on the server side. This design leads to heavyweight client-side applications that tend to be tied to a particular OS and can be difficult to deploy and support. In a three-tier architecture, the client is designed to be as lightweight as possible, normally handling only the user interface. Such a thin client might consist of forms designed in Visual Basic or perhaps only of HTML pages de cxvbgfsigned to run in a Web browser.

Business logic

While the client-server architecture is relatively fixed on deploying the client-side and the server-side components on different computers, the business logic component of a three-tier design can lead to more flexible applications. For example, the business logic of an application might be implemented as an in-process COM+ component designed to run in the process of the client application on the client side or in the process of a Web server on the server side. Alternatively, the business logic component might run in the COM+ environment on a third machine that is separate from both the client and the database server.

Editor: Bertrand Meyer, EiffelSoft, ISE Bldg., 2nd Fl., 270 Storke Rd., Goleta, CA 93117; voice (805) 685-6869; ot-column@eiffel.com

Data

The data tier of the Windows DNA model consists of SQL servers such as SQL Server 7, Oracle, Sybase, DB2, or any other database server that supports an OLE DB or the Open Database Connectivity (ODBC) specification. Typically, COM+ components running in the middle tier use ActiveX Data Objects to connect with and query the database.

DNA'S COMPONENT SERVICES

Microsoft found that developers spend too much of their time—as much as 30 percent of the total time they spend building components—writing housekeeping code. The COM+ component services at the heart of Windows DNA provide a standard implementation of services that component developers frequently need, thereby freeing developers to concentrate on the business problem at hand.

Just-in-time activation

One major feature of COM+ is its ability to scale middle-tier components so that they can support hundreds of simultaneous clients. A client that attempts to instantiate a COM+ object running in a COM+ environment receives a reference to a context object implemented by COM+—not a reference to the component's object. Only when the client later makes a method call into the component does COM+ finally instantiate the actual object. This technique, known as just-in-time activation, lets client programs obtain references to objects that they might not intend to use immediately—without incurring unnecessary overhead.

Object pooling

To enhance the overall scalability of a distributed application, COM+ supports object pooling. When a client application releases an object that supports object pooling, instead of destroying the object, COM+ recycles it for later use by the same or another client. When a client later attempts to access the same kind of object, COM+ obtains an object from the pool if one is available. COM+ automatically instantiates a new object when the pool of recycled objects is empty. Objects that support pooling are required to restore their state to that of a newly manufactured object. An object that takes a long time to create but does not hold many resources when deactivated is a good candidate for recycling.

Load balancing

A distributed COM+ application can potentially have thousands of clients. In such cases, the just-in-time activation and object pooling features can fall short of providing the required application scalability. COM+ implements load balancing at the component level, which means that a client application requesting a specific component first contacts a load balancing router. The router contains information about a cluster of machines belonging to the distributed application and balances the workload among these servers. Once the desired object has been instantiated on one of the servers in the application cluster, the client receives a reference directly to the component on the particular server. Any future requests by the client go directly to the component.

> The goal of DNA's approach is to separate the business logic from a client-server system by moving it to a middle tier that runs on Windows 2000.

In-memory database

The in-memory database, another COM+ service, is a transient, transactional database-style cache that enhances the performance of distributed applications. Implemented as an OLE DB provider, the IMDB provides fast access to data on the local machine. Client applications use high-level data access components, such as ActiveX Data Objects, to create and access indexed, tabular data. These cached databases can be generated dynamically by the COM+ application or loaded from a persistent data store.

Queued components

Queued components—a key feature of COM+—are based on the Microsoft Message Queue Server (MSMQ) infrastructure included with Windows 2000. Using queued components, a client can execute method calls against a COM+ component even if that component is offline or otherwise unavailable. The MSMQ system records and queues the method calls and automatically replays them whenever the component becomes available.

Automatic transactions

Using COM+, you can build components that can automatically participate in a distributed transaction. While transaction processing is one of its important features, COM+ actually enlists the help of the Microsoft Distributed Transaction Coordinator (DTC) to perform the transaction management. Microsoft designed OLE Transactions, an OO, two-phase commit protocol based on COM+, and then implemented the specification in MS DTC. This service is now integrated with Windows 2000, where it is available to a wide variety of applications that require transaction management services. In addition to the OLE Transactions specification, COM+ also supports the X/Open Distributed Transaction Processing XA standard (the two-phase commit protocol defined by the X/Open DTP group).

Role-based security

The key to understanding COM+ security is to understand the simple but powerful concept of *roles*. Roles are central to the flexible, declarative security model employed by most COM+ objects. A role is a symbolic name that abstracts and identifies a logical group of users. When you deploy a COM+ object, you can create certain roles and then bind those roles to specific users and user groups. For example, a banking application might define roles and permissions for tellers and for managers. During deployment, the administrator can assign users Fred and Jane to the role of tellers and assign executive management to the role of managers. Fred and Jane can access certain components in the banking package, while executive managers can access all components.

Events

Distributed applications may use COM+ events to advertise and deliver information to other components or applications without prior knowledge of the component or application identity. Event models can be categorized as either internal or external. With internal event models, the event semantic is completely contained within the scope of the publisher and subscriber, which generally requires that the publisher and subscriber run simultaneously. In the COM+ event model, the subscriber need not contain any logic for building subscriptions. In a world where subscribers greatly outnumber publishers, this is a big advantage.

Another benefit of maintaining subscriptions outside the publisher is that the subscription's life cycle need not match that of either the publisher or the subscriber. You can build subscriptions before either the publisher or the subscriber is up and running. This type of subscription, known as a persistent subscription, allows publishers to activate subscribers prior to calling them.

Currently, the component services provided by COM+ are not extensible. In other words, only Microsoft can add new services. In the future, Microsoft intends to expose the interception model on which COM+ is based to third-party developers. This would enable developers to transparently add new runtime services to COM+, making COM+ more of a service manager than a service provider. ❖

Guy Eddon is a researcher and lecturer specializing in distributed component-based systems. He is the co-author of Inside Distributed COM *(Microsoft Press, 1998). His next book,* Inside COM+, *will be released by Microsoft Press this fall. Contact him at guyeddon@msn.com.*