COM Corner:
An Introduction To COM+

by Steve Teixeira

If all goes according to Microsoft's plans, Windows 2000 will be on store shelves before the year is over *[Latest rumours are that this is maybe unlikely, but then again you never know with the folks at Redmond! Ed]*. Not only does Windows 2000 represent the first major release of a Windows NT-based product in over three years, it also represents perhaps the largest single step forward for COM since its inception as the underpinnings of OLE 2.0. COM+ is the new version of COM that will ship with Windows 2000, and this article is intended to bring you up to speed on what COM+ is and how it affects you as a Delphi developer.

COM Who?

Before I progress any further into describing COM+, please allow me to set your mind at ease by saying this: almost everything you know about COM still applies. After all, COM definitely takes no small degree of dedication to learn well, and it would be very disheartening to have to ride the same learning curve once again. The interesting thing about COM+ is that it is not a strange, new monster, but merely some fairly nice evolutionary changes to COM, combined with the integration of some of Microsoft's COM-based services that you might already be familiar with.

In plain English, COM+ can be boiled down to this: COM with a few new features, integrated with Microsoft Transaction Server (MTS) and Microsoft Message Queue (MSMQ).

Because COM+ is based on COM and fully backward compatible with COM, you have no worries from a Delphi perspective. Delphi works just as great with COM+ as it does with COM. To be able to build optimized COM+ components, there are certainly a few fundamental additions you'll need to know about, particularly with regard to a new class of components called *configured* that I will discuss in more detail later, but it's important that you know that all the world of COM+ is available to you as a Delphi developer.

Why COM?

Why did Microsoft choose to base COM+ on COM, rather than moving in some completely different direction? A fair question, especially in the light of some of the negative comments we may all hear about COM in its skirmishes with competing technologies such as CORBA and Enterprise Java Beans (EJB) in the battlefields of the industry tabloids.

Not only is COM a good foundation to build on technologically, but the business case around COM is very compelling when you consider that COM is programming language independent, COM is supported by every major Windows development tool, and every Windows 95/98/NT user is already running COM (which puts the installed base at around 150 million users according to Microsoft). The Giga Information Group recently reported that COM is a $670 million market (not including Microsoft).

Probably the biggest negative aspect of COM is its reputation for being difficult to scale to large numbers of users involved in large numbers of transactions. In typically Microsoft fashion, a major intent of COM+ is to leverage the positives while attempting to eliminate the negatives.

We can classify COM+ features into three distinct categories: runtime, services and administration. Services make up the bulk of the new features in COM+, so we'll discuss those first.

Services

COM+ services are the things that we today consider to be add-ons to COM. Technology currently found in MTS and MSMQ, for example, make up some of the services found in COM+. Think of services as systems built by Microsoft on top of COM+ designed to somehow add value to component-based development. As I mentioned, some services, such as transactions and queued components, are present thanks to off-the-shelf technology. Consequently, if you have experience with these technologies already, you'll have a leg up as you begin to write COM+ applications. Other services, however, such as object pooling and late bound events, are things that are probably new to you and may take some getting used to.

Transactions

As the 'T' in MTS, it should be no surprise to find transactions playing a major role in COM+. COM+ implements the MTS model for transactions, which I discussed previously in Issues 45 and 46. Without transaction support, there is no way a collection of objects would be able to support a complicated business application.

For example, a transaction involving an online purchase of some item might involve the participation of several objects communicating with one or more databases to receive the request, check the inventory, debit the credit card, update the accounting ledger, and finally issue a ship order. All of these things need to happen in concert; if something goes wrong in any of these processes, the state of all objects and data needs to be rolled back to the state they were in before the entire transaction began. As you can imagine, this process of managing

transactions is even more complicated when the objects involved are spread across multiple machines.

Transactions are controlled centrally by the MS Distributed Transaction Coordinator (DTC). When a COM+ application calls for transactions, the DTC will enlist the assistance of and coordinate other software elements, including transaction managers, resource managers, and resource dispensers. Each computer participating in a transaction has a *transaction manager* that tracks transaction activity on that specific machine. Transaction managers, however, are ignorant of data, as persistent information such as database data or message queue messages are managed by a *resource manager*. A *resource dispenser* manages non-persistent state information, such as database connections. Each of these specialized elements managed by the DTC know how to commit and recover its specific resource.

Security

Security is often the first thing we see in the morning. Our PCs insist that we log in before using them, and any applications we run that deal with non-trivial data often do the same. As developers, we have to know who is using the system in order to determine what type of access they will have to an application's resources. This is generally a pretty simple task on a single machine or with a single database, but as we throw distributed objects into the mix, dealing with security efficiently and effectively is not a trivial task.

While COM+ provides several approaches to security, the primary mode of security control is the role-based security system inherited from MTS. Role-based security enables an administrator to classify users as belong to specific groups, called *roles*, and provides the COM+ developer with the ability to easily check the role of a caller programmatically. This gives the developer the flexibility to do security checking where and when appropriate to the application, even on a per-method basis.

Just-In-Time Activation

Just-In-Time (JIT) activation refers to functionality already present in MTS that enables an object to be transparently destroyed and re-created without the knowledge of the client application. JIT activation potentially enables a server to handle a higher volume of clients, since resources used by an object can be reclaimed by the system when it is deactivated.

The object developer has full control over when an object is deactivated, and objects should only be deactivated when they have no state to maintain. An object can be deactivated using the `SetComplete` or `SetAbort` methods of `IObjectContext`, or the `SetDeactivateOnReturn` method of `IContextState`.

Queued Components

Queued components are based on MSMQ technology. This feature effectively allows objects to be manipulated asynchronously. An example of this is the disconnected model, when a client doesn't have a physical connection to the server but still needs to communicate some information to the server. In this model, the communication is 'recorded' while the client is offline, and 'played back' to the server object at a later time when the client is online and synchronizes.

The architecture involved in queued components involves four important pieces. First is the *recorder*, that listens to and captures the clients' attempts to manipulate the server. Second is the *queue*, which stores the 'messages' for later playback. Third is the *listener*, which removes and interprets a message from the queue, and passes it on to the player. The *player* acts as proxy for the original sender, manipulating the server object directly.

Object Pooling

You may remember that wacky `CanBePooled` method of `IObjectControl` that MTS simply ignored. The good news is that `CanBePooled` is no longer ignored, and COM+ does support object pooling. Object pooling provides the ability to keep a pool of some particular number of instances of a particular object, and having the objects in this pool be used by multiple clients. Similar to JIT activation, the goal is to increase overall throughput of the system. However, JIT activation carries the assumption that objects are not expensive to create or destroy (because it is done frequently). If an object is expensive to create or destroy, it makes more sense to keep instances around after their creation by pooling them.

There are a number of limitations imposed on objects that wish to support pooling. These include the following.

First, the object must be stateless, so it maintains no instance-specific data between method calls.

Next, the object must have no thread affinity. That is, they should not be bound to any particular thread and they should not use thread local storage (TLS, or 'threadvar' variables in the Delphi world).

The object must also be aggregatable. Resources must be manually enlisted in transactions. The resource manager cannot automatically enlist resources on the object's behalf.

Finally, the object must implement IObjectControl.

Events

Delphi developers don't need to be sold on the importance of events. How else would we know when a button was clicked or a record posted? However, while COM developers have also been aware of the importance of events, they often avoided them due to the complexity of implementation. COM+ introduces a new event model, which, thank heavens, is not tied to the Byzantine *connection points* model that has been common in COM to this point.

The COM+ event model is based on an asynchronous publish-and-subscribe model that functions on

a per-method-level. This is a huge step forward for two reasons: first, events are queued and fired asynchronously by the system, meaning that the source object can 'fire and forget' and know that COM+ will handle the propagation. Second, the connection points implementation of events required that event listeners implement an entire interface, even if they were only interested in one method, whereas events are now single methods. Also present in the COM+ event model is the ability to filter events based on parameter values, so that you can be very specific about the events you're interested in.

Services Not In COM+

Beta releases and early documentation on COM+ included some services that it appears will not be in the final release of Windows 2000. Being beta software, it is all of course subject to change, but based on the latest information available from Microsoft, the following services will not be present in the shipping version of COM+: load balancing, the in-memory database, and the transactional shared property manager.

Load Balancing

Load balancing increases the scalability of a system by enabling workload to be distributed across multiple machines. The COM+ implementation of load balancing was known as Component Load Balancing (CLB). Microsoft has chosen to remove CLB from Windows 2000 and instead offer it as a separate add-on available in the upcoming Microsoft AppCenter Server.

In-Memory Database (IMDB)

Beta versions of Windows 2000 included a feature called IMDB, which was effectively an in-memory, transactional cache that operated on database semantics, enabling very fast storage and access to data. Recent statements from Microsoft indicate that the IMDB will not ship in Windows 2000 because it does not fully address users' needs (notably query processing and stored procedures).

Transactional Shared Property Manager (TSPM)

The TSPM was based on IMDB, and therefore will also not be found in Windows 2000. COM+ will instead incorporate the shared property manager technology that is currently found in MTS.

Runtime

You can think of the COM+ runtime as essentially the COM you already know and love. The COM+ runtime is comprised of all the various COM API functions (all those functions starting with `Co...`) and the underlying code that makes those functions go. The runtime handles things like object creation and lifetime, marshalling, proxies, memory management, and all the other low-level things that make up the foundation of COM+. In order to support many of the nifty services you just learned about, Microsoft has added a number of new features to the COM+ runtime, including configured components, a registration database, the promotion of the contexts concept, and a new 'neutral' threading model.

Registration Database (RegDB)

In COM, the attributes of a particular COM object are generally kept in two places: the system registry and a type library. COM+ now introduces the concept of a registration database that will be used to hold attribute information for COM+ object. Type libraries will continue to be used, but the system registry has distinctly fallen out of favour as the place to store object attributes, and use of the registry for this purpose is supported only for the sake of backward compatibility. Common attributes stored in the RegDB include the transaction level supported by an object and whether it supports JIT activation.

Configured Components

Components that store attributes in RegDB are referred to as *configured components*, whereas components that do not are called *non-configured*. The best example of a non-configured component is a COM or MTS component that you are using unchanged in the COM+ environment. To participate in most of the services I mentioned earlier, your components will need to be configured.

Contexts

Contexts are a term originally introduced in MTS that described the state of the current execution environment of a given component. Not only has this term moved forward in COM+, but it has been promoted. In COM, an apartment is the most granular description of the runtime context of a given object, referring to an execution context bounded by a thread or process. In COM+ that honour goes to a context, which runs within some particular apartment. A context implies description on a more granular level than an apartment, such as transaction and activation state.

Neutral Threading

COM+ introduces a new threading model, known as Thread Neutral Apartment (TNA). TNA is designed to provide the performance and scalability benefits of a free threaded object without the programming problems of dealing with interlocking access to shared data and resources within the server. TNA is the preferred threading model for COM+ components that do not surface UI elements. Components containing UI should continue to use apartment threading, since window handles are tied to a specific thread. There is a limitation of one TNA per process.

Administration

In order to help manage all of these new runtime features and services, Microsoft provides a couple of different means for management and administration of the COM+ environment. The first is a Microsoft Management Console (MMC) snap-in that enables you to manipulate COM+ component attributes in the RegDB. The

second is a set of APIs collectively known as the Component Services Administration (COMAdmin) Library that enable you to write code to perform such as tasks as: Creation, installation, and configuration of COM+ applications; management of installed COM+ applications; management and configuration of COM+ services; remote administration Component Services on a different machine.

Summary

That sums up this introduction to COM+, the next version of COM that will ship with Windows 2000. It's a very exciting time to be a COM developer, with all this great new technology coming just down the road. Remember, the motivation behind all of this is to provide COM developers the ability to build truly scalable applications. As you design that next web-based or multi-tier system, remember to keep COM+ features things in mind. As the months progress, this column will drill down into implementations of many of the technologies you see here today.

Steve Teixeira is the VP of software development at DeVries Data Systems and co-author of the upcoming *Delphi 5 Developer's Guide*. You can reach Steve at steve@dvdata.com