

## COM+ Application Patterns

Article 2 of 8 in COM+ Design Patterns series

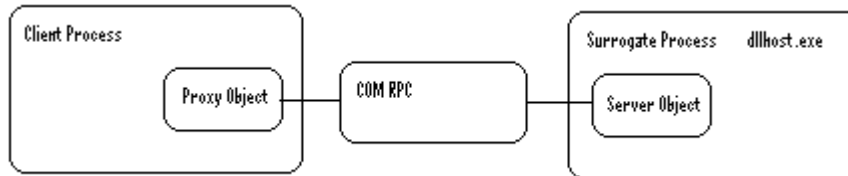
by Randy Charles Morin

As I discussed in the first article in this series, frameworks like COM evolve by adopting a larger range of features. One set of features adopted in COM+ is the range of application design patterns. COM+ Applications were adopted from the deprecated MTS (Microsoft Transaction Service). As with MTS Applications, the central design pattern that Microsoft was trying to replicate is the surrogate design.

### The Surrogate Design Pattern

The Surrogate Design Pattern is very similar in concept to surrogate mothering. That is, in a surrogate pregnancy, one woman carries the fetus on behalf of other people. In COM+, a surrogate process houses objects on behalf of client processes.

Figure 1: Surrogate Process



This is similar to the client-server model presented by COM/DCOM. So you might ask what is new here? The advantage of the surrogate design pattern is that objects can easily be ported between different surrogate processes on the same machine or on different machines. These surrogate processes can each have different qualities that are acquired by the housed objects.

In the standard COM/DCOM client-server model, the executable code for server classes exist in the server EXE (executable). In the surrogate client-server model, the executable code for the server classes exist in DLLs (dynamic link libraries). These DLLs can be loaded into any surrogate process and even the client process.

The advantage means that developers can load DLL classes in process or in a fixed server for ease of development and debugging. While in production, the DLL classes can be loaded in the surrogate processes on one or many machines. The ability to locate your objects on more than one machine means you have an almost limitless ability to scale your application. If one computer can handle a hundred thousand users, then ten computers should be able to handle nearly a million users.

The COM+ Surrogate is a merging of the COM Default Surrogate (DLLHOST.EXE) that was deployed with the NT 4 SP2 (Service Pack Two) and the MTS Surrogate (MTX.EXE). The COM Default Surrogate made it possible to load COM components contained within a DLL into external processes. Previous to the COM Default Surrogate, COM components contained within a DLL could only be loaded in-process and COM components contained within a EXE could only be loaded out-of-process or remotely.

Using the COM Default Surrogate it was then possible to load COM components contained within a DLL, in the local process, out-of-process and remotely.

The MTS Surrogate had additional benefits in that it provided for object pooling, late activation and more. When COM components were loaded into the MTS surrogate process, the proxy object was not bound to a server object. This removed a network cycle during calls to create an object instance and allowed the MTS run-time to dynamically bind proxy objects to server objects. This dynamic binding allowed the MTS run-time to support a larger amount of proxy objects with a smaller amount of server objects.

Supporting a large amount of proxy objects with fewer server objects is very important in increasing the scalability of an application. If ten thousand proxy objects represented ten thousand concurrent users, then normal COM applications would require ten thousand server objects.

#### Note

This is not necessarily true that normal COM applications would require an equal amount of proxy and server objects. Many programmers use a singleton COM class where calls to the `IClassFactory::CreateInstance` method always return the same instance. This means that many proxy objects are using the same server object instance. Most COM theologians consider this singleton pattern to be a violation of the COM specification.

In an MTS application (or COM+), you could support ten thousand concurrent users with ten thousand proxy objects and only a fraction of the server objects. Assuming the ratio is one server object to one thousand proxy objects and that your typical application server could support a thousand simultaneous server objects, then your normal COM application (without MTS or COM+) would require one application server for every thousand concurrent users.

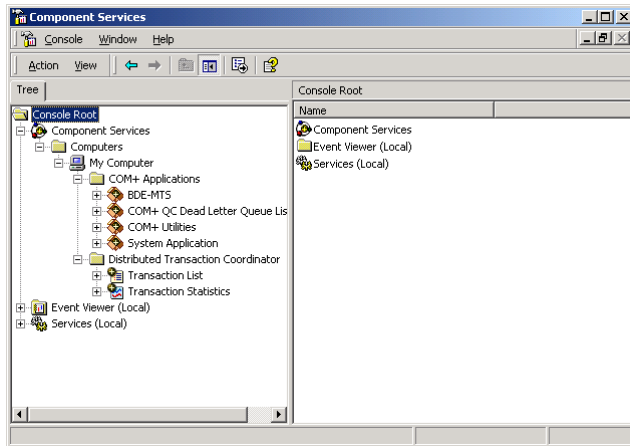
A similar MTS (or COM+) application on the other hand could support a million concurrent users per application server. Now that's distributed objects that scale.

The COM+ Applications provide us with another major advantage. COM+ applications can be managed from a common administrative framework, the COM+ Catalog.

## The Centralized Management Design Pattern

The Centralized Management Design Pattern fell out of the MMC (Microsoft Management Console) initiative from a few years past. The basic idea is to co-locate all administrative tools in one common GUI. From the MMC, you can administer a large range of applications that were previously administered from many unique administration GUIs. Currently MMC has support for COM+, SQL Server 7, the Event Viewer, Windows Services and much more. You can also create your own MMC snap-ins and administer pretty much whatever you want from this common GUI.

In MTS, the MMC (Microsoft Management Console) snap-in used to administer MTS Applications was called MTS Explorer. The equivalent utility in COM+ is called Component Services.

**Figure 2: Component Services**

The Component Services is the GUI (Graphic User Interface) front-end for the COM+ Catalog. The COM+ Catalog is the logical database where COM+ configurations are stored. In both COM and DCOM, configurations were stored in the Windows Registry.

The COM+ Catalog is a hierarchical configuration structure used to configure COM+ applications-servers. It is very similar to the Windows Registry in that it organizes configuration data in this hierarchical structure.

Assuming you are familiar with the Windows Registry, you'll remember that it is divided into two major hives, the HKEY\_LOCAL\_MACHINE and HKEY\_USERS.

It should be noted, that the COM+ Catalog is a logical data store. The data in the COM+ Catalog is persisted in two physical data stores, the Windows Registry under HKEY\_CLASSES\_ROOT and in the COM+ Registration Database.

Access to the COM+ Catalog is provided through a COM class called the COMAdminCatalog. Listing 1 shows a simple Visual Basic Script (VBS) sample of accessing and navigating this administrative class.

#### **Listing 1: Accessing the COM+ Catalog**

```
Dim catalog
Dim collection
Dim item

'Equivalent of newline
NewLine = Chr(13) + Chr(10)

'Navigate the COM+ Catalog
set catalog = CreateObject("COMAdmin.COMAdminCatalog")
set collection=catalog.GetCollection(WScript.Arguments(0))
collection.Populate()
num = collection.Count

'Header
output = "Installed COM+ Applications" & NewLine & "======"
WScript.Echo output

'View Collection
For I = num - 1 to 0 step -1
set item = collection.Item(I)

output = item.Name
WScript.Echo output
```

Next

```
output = NewLine & NewLine
WScript.Echo output

set catalog=Nothing
set collection=Nothing
set item=Nothing
```

Note

In order to run any VBS samples in this article or any other articles, you can copy the contents into a file with a VBS extension. Then open the file and the VBS Engine will run the script.

## Application Design Pattern

The COM+ Catalog is organized into top-level collections; Applications, ComputerList, ApplicationCluster, ErrorInfo, LocalComputer, InprocServers, DCOMProtocols, TransientSubscriptions, Root, RelatedCollectionInfo and PropertyInfo. The most important collection of the COM+ Catalog is the Applications collection. Under the Application collection you configure nearly everything about a COM+ Application. This includes the components that are part of the COM+ Application, the security roles that exist in the COM+ Application and the persistent subscriptions in the COM+ Application. I'll explain security roles and persistent subscriptions in later articles in this series called COM+ Security Patterns and COM+ Publication Patterns.

The COM+ Catalog can be accessed, navigated and updated using three interfaces, ICOMAdminCatalog, ICatalogCollection and ICatalogObject.

### Listing 2: COM+ Catalog Interfaces

```
interface ICOMAdminCatalog : IDispatch {
    HRESULT GetCollection(
        [in] BSTR bstrCollName,
        [out, retval] IDispatch** ppCatalogCollection);
    HRESULT ShutdownApplication([in] BSTR bstrApplIdOrName);
    HRESULT StartApplication([in] BSTR bstrApplIdOrName);
};

interface ICatalogCollection : IDispatch
{
    HRESULT Item( [in] long lIndex,
        [out, retval] IDispatch** ppCatalogObject);
    HRESULT Count([out, retval] long* retval);
    HRESULT Remove([in] long lIndex);
    HRESULT Add([out, retval] IDispatch** ppCatalogObject);
    HRESULT Populate();
    HRESULT SaveChanges([out, retval] long* retval);
    HRESULT GetCollection(
        [in] BSTR bstrCollName,
        [in] VARIANT varObjectKey,
        [out, retval] IDispatch** ppCatalogCollection);
    HRESULT Name([out, retval] VARIANT* retval);
};

interface ICatalogObject : IDispatch {
    HRESULT Value(
        [in] BSTR bstrPropName,
        [out, retval] VARIANT* retval);
    HRESULT Value(
        [in] BSTR bstrPropName,
```

```
        [in] VARIANT retval);  
    HRESULT Key([out, retval] VARIANT* retval);  
    HRESULT Name([out, retval] VARIANT* retval);  
};
```

The interfaces were reduced in order to make them more presentable in the article. The interfaces have two or more times as many methods.

You can think of the COM+ Catalog as a tree structure, with a root node, branches of parent nodes and leaves of children nodes. The ICOMAdminCatalog interface is used to navigate the root, the ICatalogCollection interface is used to navigate the parent nodes and the ICatalogObject interface is used to navigate the children nodes.

Navigating the COM+ Catalog is a bit more complex than I originally hoped. The following listing shows how to navigate to a COM+ application and enumerate the components that are installed in the application.

### Listing 3: Navigating the COM+ Catalog

```
Dim catalog  
Dim collection  
Dim item  
Dim object  
Dim components  
  
'Equivalent of newline  
NewLine = Chr(13) + Chr(10)  
  
'Navigate the COM+ Catalog  
set catalog = CreateObject("COMAdmin.COMAdminCatalog")  
set collection = catalog.GetCollection("Applications")  
collection.Populate  
  
num = collection.Count  
  
'View Collection  
For I = num - 1 to 0 step -1  
set item = collection.Item(I)  
  
if item.Name = WScript.Arguments(0) then set object = item  
  
Next  
  
'Navigating collection within components is a bit weird.  
'You specify the call in form root.GetCollection(collectionname, objectkey)  
set components = collection.GetCollection("Components", object.key)  
components.Populate  
  
num = components.Count  
  
'View Collection  
For I = num - 1 to 0 step -1  
set item = components.Item(I)  
  
output = output & item.Name & " "  
  
Next  
  
WScript.echo output  
  
set catalog = Nothing  
set collection = Nothing  
set item = Nothing  
set object = Nothing  
set components = Nothing
```

The tricky part is where you try to get the collections inside of other objects. You re-use the root collection object and ask for the collection name for a particular object. That is, the first parameter in the method call is the collection name and the second is the key of the branch object (root.GetCollection collectionname branchobjectkey). The key of a branch object is a secondary representation of the branch, but in GUID (Global Unique Identifier) form.

Running the previous script with one parameter, the name of the COM+ application, will output the name of all components within that application.

## Object Context Design Pattern

Most of the new features in COM+, many inherited from MTS, are implemented using a COM+ objects' context. This context is provided to COM+ objects by associating a context object with each COM+ object. The context object can be acquired using the CoGetObjectContext function. This function is used to acquire an interface to the context object.

In MTS, the GetObjectContext function was provided that returned an IObjectContext interface to the context object. The new CoGetObjectContext function provided additional interfaces that are support by the context class. Where as the GetObjectContext function returned the IObjectContext interface, the CoGetObjectContext function can also return an interface pointer to the IObjectContextInfo, IContextState and ISecurityCallContext interfaces.

The IObjectContext interface is a legacy interface left over from MTS days when COM and MTS were not integrated and you were required to explicitly use the context. The COM+ run-time will automatically use the object's context, without requiring additional coding.

### Listing 4: IObjectContext IDL

```
interface IObjectContext : IUnknown {
    HRESULT CreateInstance(
        [in] GUID* rclsid,
        [in] GUID* riid,
        [out, retval] void** ppv);
    HRESULT SetComplete();
    HRESULT SetAbort();
    HRESULT EnableCommit();
    HRESULT DisableCommit();
    long IsInTransaction();
    long IsSecurityEnabled();
    HRESULT IsCallerInRole(
        [in] BSTR bstrRole,
        [out, retval] long* pfIsInRole);
};
```

If you wanted to create another COM object and have that object use the same object context, then you can create the object with the CreateInstance method in the context object. SetComplete and SetAbort allow voting on whether transaction associated with the context object should be rolled back or committed.

The methods of the IObjectContext interface provide coarse-grained control of the transaction, activation and security of a context object. The IObjectContextInfo,

IContextState and ISecurityCallContext interfaces provide fine-grained control over the transaction, activation and security of a context object.

This is the second article in a series of eight on COM+ Design Patterns. The third and next article in this series is on COM+ transaction patterns.

## About the Author

Randy Charles Morin is the Lead Architect of SportMarkets Development from Toronto, Ontario, Canada and lives with his wife and two kids in Brampton, Ontario. He is the author of the [www.kbcafe.com](http://www.kbcafe.com) website, author of Wiley's Programming Windows Services book and co-author of many other programming books. Feedback from Bruce Duffus was used in writing this article.

## References

COM+ <http://www.microsoft.com/com/tech/complus.asp>  
Design Patterns by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, published by Addison-Wesley Professional Computing