

DCOM Bi-Directional Communication Derby

Callbacks vs Connection Points

by Randy Charles Morin

This is Randy Charles Morin reporting to you live from the DCOM racetrack. Today's race has remote DCOM callbacks against remote DCOM connection points. This is considered a non-contest as everybody knows that remote DCOM callbacks are much faster than remote DCOM connection points. Today we're here to verify how much faster these callbacks are than their connection point brothers. And now over to Selrahc Norim who be doing the color for this race.

The contestants in this race are the well-known bi-directional communication mechanism known as DCOM callbacks and DCOM connection points.

Callbacks

First an introduction to our clearly superior contestant, DCOM callbacks. DCOM callbacks are implemented by passing an interface pointer from the event sink to the event source. The IDL for a typical DCOM callback server is in listing 1.

Listing 1: IDL for Callback Server

```
[
    object,
    uuid(606DE2C5-1A8C-11D3-B68F-00C04F8B72E7),
    dual,
    helpstring("ICallback Interface"),
    pointer_default(unique)
]
interface ICallback : IDispatch
{
    [id(1), helpstring("method Advise")]
    HRESULT Advise([in] IUnknown * punknow, [out] DWORD * dwCookie);
    [id(2), helpstring("method Unadvise")]
    HRESULT Unadvise([in] DWORD dwCookie);
};
[
    object,
    uuid(73296190-1A8F-11D3-B68F-00C04F8B72E7),
    dual,
    helpstring("_ICallbackEvents Interface"),
    pointer_default(unique)
]
interface _ICallbackEvents : IDispatch
{
    [id(1), helpstring("method Hello")] HRESULT Hello();
};
[
    uuid(606DE2C6-1A8C-11D3-B68F-00C04F8B72E7),
    helpstring("Callback Class")
]
coclass Callback
{
    [default] interface ICallback;
};
```

Note that the ICallback interface implements Advise and Unadvise methods to begin and terminate callback notification to a client. The client can simply query the ICallback interface and pass a client object that implements the _ICallbackEvents interface to

initiate the bi-directional communication. This is quite simple and should be very fast. DCOM connection points will have a difficult time replicating such simplicity and speed.

The IDL for a typical DCOM callback client is in listing 2.

Listing 2: IDL for Callback Client

```
importlib("../connectionpointserver/connectionpointserver.tlb");
[
    uuid(F67DB88D-1A8C-11D3-B68F-00C04F8B72E7),
    helpstring("CallbackClient Class")
]
coclass CallbackClient
{
    [default] interface _ICallbackEvents;
};
```

Here again is the simplicity of the callback client. It simply implements the event interface provided by the server and registers this callback object with the callback server. When the event is triggered the "Hello" method is called and the client callback code is executed. This simple interface and implementation should prove very hard to beat for the DCOM connection point contestant.

Connection Points

Our second contestant is no other than DCOM connection points. As we have all been told in the past, DCOM connection points are slow and should be avoided when used in a distributed environment. Our secondary contestant who is bound to lose has the following IDL for its typical server.

Listing 3: IDL for Connection Point Server

```
[
    object,
    uuid(606DE2C1-1A8C-11D3-B68F-00C04F8B72E7),
    dual,
    helpstring("_ISlowConnectionPointsEvents Interface"),
    pointer_default(unique)
]
interface _ISlowConnectionPointsEvents : IDispatch
{
    [id(1), helpstring("method Hello")] HRESULT Hello();
};
[
    object,
    uuid(606DE2BF-1A8C-11D3-B68F-00C04F8B72E7),
    dual,
    helpstring("ISlowConnectionPoints Interface"),
    pointer_default(unique)
]
interface ISlowConnectionPoints : IDispatch
{
};
[
    uuid(606DE2C0-1A8C-11D3-B68F-00C04F8B72E7),
    helpstring("SlowConnectionPoints Class")
]
coclass SlowConnectionPoints
{
    [default] interface ISlowConnectionPoints;
    [default, source] interface _ISlowConnectionPointsEvents;
};
```

Note this server uses the source attribute in the coclass interface definition. This means the server must implement the very slow IConnectionPointContainer and IConnectionPoint interfaces. This server is going to need a lot of luck to pull off an upset in this contest. Of course the advantage of using connection points is that ATL provides all the boilerplate implementation and nothing additional needs to be coded. Callbacks don't have as much supporting code so they do require some additional coding in the server.

The DCOM connection point client is similar to the DCOM callback client. Here's the IDL for our DCOM connection points client.

Listing 4: IDL for Connection Point Client

```
importlib("..\\connectionpointserver\\connectionpointserver.tlb");
[
    uuid(F67DB889-1A8C-11D3-B68F-00C04F8B72E7),
    helpstring("SlowClient Class")
]
coclass SlowClient
{
    [default] interface _ISlowConnectionPointsEvents;
};
```

Because of the similarity in the client IDL, it is pretty much guaranteed that the triggering of events will occur at identical speeds. So we don't have to worry about testing this part of the bi-directional communication mechanism.

The Test

The important part to test is the speed at which we are able to establish bi-directional communication. We have therefor established a test where the bi-directional communication is established and broken repeatedly one thousand times. The listing for the test follows.

Listing 5: Test Code

```
// testclient.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include "ComObjectPtr.h"
#include "..\\connectionpointserver\\connectionpointserver.h"
#include "..\\connectionpointserver\\connectionpointserver_i.c"
#include "..\\connectionpointclient\\connectionpointclient.h"
#include "..\\connectionpointclient\\connectionpointclient_i.c"
#define CYCLES 1000
int connectionpoints()
{
    DWORD dw = ::GetTickCount();
    for (int i=0;i<CYCLES;i++)
    {
        HRESULT hresult;
        // Initiate the slow client
        ComObjectPtr<IUnknown> slowclient;
        ComObjectPtr<IConnectionPointContainer> slowserver;
        slowclient.CreateObject(CLSID_SlowClient, IID_IUnknown);
        slowserver.CreateObject(CLSID_SlowConnectionPoints,
            IID_IConnectionPointContainer);
        DWORD slowcookie;
        IConnectionPoint * pcp = NULL;
        hresult = slowserver->FindConnectionPoint(
            IID__ISlowConnectionPointsEvents, &pcp);
        if (FAILED(hresult))
        {
            ::OutputDebugString("FindConnectionPoint failed in "
```

```

        "connectionpoints");
    }
    hresult = pcp->Advise(slowclient, &slowcookie);
    if (FAILED(hresult))
    {
        ::OutputDebugString("Advise failed in connectionpoints");
    }
    // Close the slow client
    hresult = pcp->Unadvise(slowcookie);
    if (FAILED(hresult))
    {
        ::OutputDebugString("Unadvise failed in connectionpoints");
    }
    pcp->Release();
}
std::stringstream ss;
ss << "Elapsed milliseconds = " << ::GetTickCount() - dw << "\n"
    << "Cycles = " << CYCLES << "\n"
    << "Ran slow client\n\n";
::OutputDebugString(ss.str().c_str());
return 0;
}

int callbacks()
{
    DWORD dw = ::GetTickCount();
    for (int i=0;i<CYCLES;i++)
    {
        HRESULT hresult;
        // Initiate the call client
        ComObjectPtr<IUnknown> callbackclient;
        ComObjectPtr<ICallback> callbackserver;
        callbackclient.CreateObject(CLSID_CallbackClient, IID_IUnknown);
        callbackserver.CreateObject(CLSID_Callback, IID_ICallback);
        DWORD callbackcookie;
        hresult = callbackserver->Advise(callbackclient, &callbackcookie);
        if (FAILED(hresult))
        {
            ::OutputDebugString("Advise failed in callbacks");
        }
        // Close the callback client
        hresult = callbackserver->Unadvise(callbackcookie);
        if (FAILED(hresult))
        {
            ::OutputDebugString("Unadvise failed in callbacks");
        }
    }
    std::stringstream ss;
    ss << "Elapsed milliseconds = " << ::GetTickCount() - dw << "\n"
        << "Cycles = " << CYCLES << "\n"
        << "Ran callback client\n\n";
    ::OutputDebugString(ss.str().c_str());
    return 0;
}

int main(int argc, char* argv[])
{
    ::CoInitializeEx(NULL, COINIT_MULTITHREADED);
    connectionpoints();
    callbacks();
    connectionpoints();
    callbacks();
    connectionpoints();
    callbacks();
    ::CoUninitialize();
    return 0;
}

```

In order to make the test as fair as possible, we've decided to run the test five times over and take the second thru fourth set of results. This eliminates any unfair advantage one

server might have over the other in restarting-loading the bi-directional communication server component.

The contestants have agreed to start with a small test using local (not remote) communication to give them some time to tweak their engine speeds. Our first run will be using a client on the same computer as the server. Later we will run the real contest where the client and server are located on separate machines.

There off, connection points get off to a big lead. This is amazing, how can connection points compete. Clearly the read literature says that connection points should not be able to compete with callbacks. Surely callbacks will make a late charge and defeat the foe. And at the wire, it's connection points by a large margin. There obviously must be a problem with the callback server. Something is wrong in the callback camp.

Listing 6: The Local Server Results

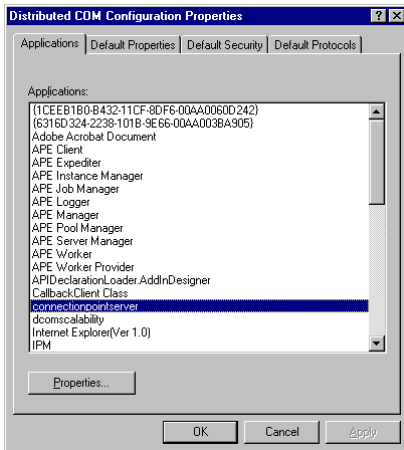
```
Elapsed milliseconds = 6219
Cycles = 1000
Ran slow client
Elapsed milliseconds = 7471
Cycles = 1000
Ran callback client
Elapsed milliseconds = 5498
Cycles = 1000
Ran slow client
callback called
Elapsed milliseconds = 7020
Cycles = 1000
Ran callback client
Elapsed milliseconds = 5228
Cycles = 1000
Ran slow client
Elapsed milliseconds = 7010
Cycles = 1000
Ran callback client
Elapsed milliseconds = 5327
Cycles = 1000
Ran slow client
Elapsed milliseconds = 6920
Cycles = 1000
Ran callback client
Elapsed milliseconds = 5578
Cycles = 1000
Ran slow client
Elapsed milliseconds = 7271
Cycles = 1000
Ran callback client
```

Amazingly the connection point framework was 30% faster than its callback brother. But as everybody knows, connection points were design to operate efficient in a very local setup. Surely callbacks are still in for a win in the remote marathon race.

The Marathon

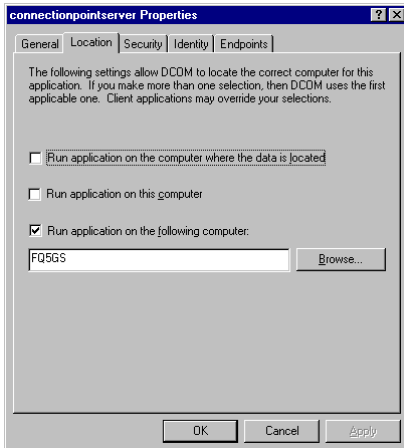
Now let's reconfigure the server to run as a remote component. We can do this using DCOMCNFG (see figure 1). You can start DCOMCNFG by select Start | Run from the taskbar, then type DCOMCNFG and click OK.

Figure 1: DCOMCNFG



Select connectionpointserver and click on the Properties button. Now select the Location tab (see figure 2). You can configure a server to run on a remote machine by unchecking Run application on this computer and checking Run application on the following computer. Select the remote computer name using the Browse button and dialog.

Figure 2: DCOMCNFG Location Properties



Don't forget also to install the server component on the remote server machine by copying the server executable and executing the connectionpointserver /regserver command-line.

Now we're ready for the ultimate test. Of course this isn't much of a test as conventional wisdom tells us that connection points are much slower than callbacks. And both contestants are now ready in the starting blocks. On your marks! Get Ready! Go!

Wait a second connection points are off to another great start. But this time, the lead is not as much as the last. Callbacks are right on their heels. And at the wire, it's connection points. Huh??? Connection points finished 15% faster than callbacks.

Listing 7: The Remote Server Results

```
Elapsed milliseconds = 10365
Cycles = 1000
Ran slow client
Elapsed milliseconds = 9744
```

```
Cycles = 1000
Ran callback client
Elapsed milliseconds = 8583
Cycles = 1000
Ran slow client
Elapsed milliseconds = 9673
Cycles = 1000
Ran callback client
Elapsed milliseconds = 8923
Cycles = 1000
Ran slow client
Elapsed milliseconds = 10385
Cycles = 1000
Ran callback client
Elapsed milliseconds = 9003
Cycles = 1000
Ran slow client
Elapsed milliseconds = 10485
Cycles = 1000
Ran callback client
Elapsed milliseconds = 8993
Cycles = 1000
Ran slow client
Elapsed milliseconds = 9924
Cycles = 1000
Ran callback client
```

Connection Points Win

You may have read several books and articles where authors tell the reader not to use connection points because they are too slow. I've heard this same argument made by every single expert in the DCOM business. But in every case, the discussion remained theoretical and nobody bothered to produce empirical evidence. So, I decided having had good experiences with remote connection points to test the theory with a practical implementation. These tests were conducted on a couple of SP4 machines. The results always favored connection points. Different configurations may produce other results, but I haven't tested them.

But let us return to the theoretical. Why do connection points out-perform callbacks in this test? I think the biggest clues will be made public by publishing the wire trace of frames passing between the client and server components.

About the Author

Randy Charles Morin is co-author of [COM/DCOM Unleashed](#) and [COM/DCOM Primer Plus](#).