C# COM Interoperability Late Binding

Vijay Appadurai
with small revisions by Jim Fawcett

CSE775 – Distributed Objects

Spring 2005

Types of Binding

- There are two kinds of Binding from C# to COM – Early Binding and Late Binding.
- Early Binding can be done by creating a runtime callable wrapper, which the C# client can use for invoking COM objects. That's what happens when you make a reference in a C# client to a COM server.
- Late Binding can be done even without the creation of a runtime callable wrapper. We will see how.

Late Binding

- Late Binding is done with the help of the C# Reflection APIs.
- The Type class and the Activator class of the C# Reflection API is used for this purpose.
- The C# client only needs to know the server's Program ID for runtime invocation. The following code shows how to accomplish that.

Using C# Reflection for Late Binding

```
//Get IDispatch Interface from the COM Server. Here the Server's Program ID is
    "Component.InsideDCOM"
Type objType =
    Type.GetTypeFromProgID("Component.InsideDCOM");

//Create an instance of the COM object from the type obtained
object objSum = Activator.CreateInstance(objType);

object c;
object[] myArgument = {100,200};

//Invoke a Method on the COM Server which implements IDispatch Interface and get
    the result
c = objType.InvokeMember("Sum",
    BindingFlags.InvokeMethod, null, objSum, myArgument);

//Print the result
Console.WriteLine("Sum of 100 and 200 is " + c);
```

Making COM Server Support Late Binding

- To support Late Binding, the COM Server should implement the IDispatch Interface.
- This can be done in two ways:
- **THE PURE AUTOMATION INTERFACE**

  Use the dispinterface statement shown here when you are designing a pure automation interface:

```
 [uuid(10000001 – 0000 – 0000 – 0000 - 000000000001)]
dispinterface ISum
{
      properties:
       methods:
             [ id(1)] int Sum(int x, int y);
};
```

Dual Interfaces

- Using the dispinterface is not recommended since doing so restricts a client to using only the IDispatch interface.
- Making dual interfaces is preferred.
- Here's the IDL syntax required to indicate support for both IDispatch and custom interface.

```
[object, uuid(10000001 – 0000 – 0000 – 0000 -000000000001),
 dual]
interface ISum : IDispatch
{
        [id(1)] HRESULT Sum (int x, int y, [out, retval] int* retval);
}
```

Modifying Outproc3a and 3b to Support Dual Interface

- Implement all four functions of IDispatch:

```
  // IDispatch
HRESULT __stdcall GetTypeInfoCount(UINT* pCountTypeInfo);

HRESULT __stdcall GetTypeInfo(UINT iTypeInfo, LCID lcid, ITypeInfo**
   ppITypeInfo);

HRESULT __stdcall GetIDsOfNames(REFIID riid, LPOLESTR*
   rgszNames, UINT cNames, LCID lcid, DISPID* rgDispId);

HRESULT __stdcall Invoke(DISPID dispIdMember, REFIID riid, LCID
   lcid, WORD wFlags, DISPPARAMS* pDispParams, VARIANT*
   pVarResult, EXCEPINFO* pExcepInfo, UINT* puArgErr);
```

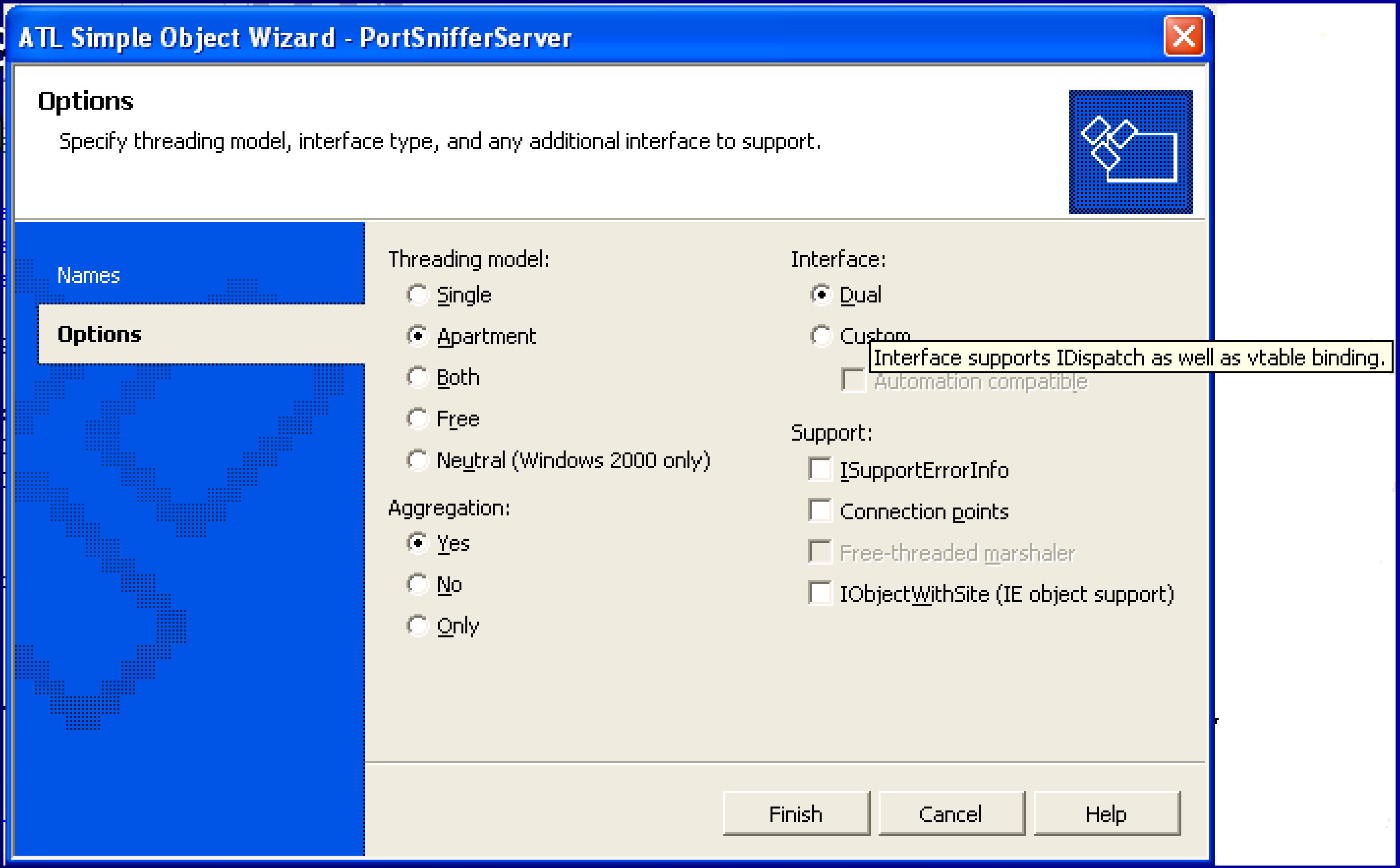
Modifying Ouproc3a and 3b

- Modify QueryInterface so that it returns IDispatch* when queried for IID_IDispatch
- Get Type Information about the ISum interface in the CFactory::CreateInstance function.

```
HRESULT
CFactory::CreateInstance(IUnknown *pUnknownOuter, REFIID riid, void
    **ppv)
   {
    …
    ITypeLib* pTypeLib;

   LoadRegTypeLib(LIBID_Component, 1, 0, LANG_NEUTRAL, &pTypeLib)

   HRESULT hr = pTypeLib->GetTypeInfoOfGuid(IID_ISum, m_pTypeInfo);

   pTypeLib->Release();
   …
   }
```

Using ATL to support Dual Interfaces

- Its
  sh



ATL Simple Object Wizard - PortSnifferServer

Options

Specify threading model, interface type, and any additional interface to support.

Names

Options

Threading model:
- Single
- [✓ Apartment]
- Both
- Free
- Neutral (Windows 2000 only)

Aggregation:
- [✓ Yes]
- No
- Only

Interface:
- [✓ Dual]
- Custom
  - Automation compatible

Interface supports IDispatch as well as vtable binding.

Support:
- ISupportErrorInfo
- Connection points
- Free-threaded marshaler
- IObjectWithSite (IE object support)

References:

Inside Distributed COM,
Guy Eddon and Henry Eddon,
Microsoft Press, 1998