# C# COM Interoperability

Handling COM Events in C#

Vijayanand Appadurai
with small revisions by Jim Fawcett
CSE775 Distributed Objects
Spring 2005, 2007

# Ways of using Connection Points in Managed Code

- There are three ways of using Connection points in managed code.
  1. Raw Connection Points Approach.
  2. Type Library Importer Transformations.
  3. .Net Reflection.

- Raw Approach isn't recommended. It does not take advantage of built-in event specific Interop support which we will now discuss.

# Type Library Importer Transformations

- To expose connection points as .Net events, the Type Library Importer does a lot of extra work.
- Every time the type library importer encounters an interface listed with the [source] attribute it creates the following types.
  1. SourceInterfaceName_Event.
  2. SourceInterfaceName_MethodNameEventHandler.
  3. SourceInterfaceName_EventProvider.
  4. SourceInterfaceName_SinkHelper.
- These types expose the COM connection points as standard .Net events which can be viewed using the object browser.
- Just hook up event handlers for the events we care about and it works.

# Handling Events using .Net Reflection

- The Type Library Importer Transformation approach can be used only when we use early binding to the server.
- To handle server events using late binding we need to use the .Net Framework API's directly instead of using the .Net Framework SDK's.
- Now we don't have to add a reference to the server. Just the server GUID will suffice. We will see how this works.

# Handling COM Events in C#

- Step1: Get the path of the component from the registry.
- Step2: Load the server's type library, using the server path obtained in the previous step.
- Step3: Generate a .Net assembly at runtime from the type library object obtained in Step2.
- Step4: Reflect on the assembly to get the server events we care about and add event handlers to it at runtime.

# Getting Component Path from Registry

- The component path can be obtained from the registry using the .Net Registry API's.
- The path of the component is stored in the registry under **HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\ {GUID}\LOCALSERVER.**
- Using the GUID, search the registry for the above key and get the component path.

# Load the server's type library

- To generate a .Net assembly we need an object that implements the ITypeLib interface as input.
- To get such an interface we can use the OLE Automation LoadTypeLibEx method. This can be done using PInvoke:
- ```
  [ DllImport( "oleaut32.dll", CharSet = CharSet.Unicode, PreserveSig = false )]
      private static extern void LoadTypeLibEx(
        String strTypeLibName, RegKind regKind,
        [ MarshalAs( UnmanagedType.Interface )] out Object typeLib
  );
  ```
- This can be then used in C# as follows.

  **Object typeLib;**

  **LoadTypeLibEx( path, RegKind.RegKind_None, out typeLib);**
  ```

# Generate a .Net assembly at runtime

- The process of generating a .Net assembly from a type library can be done using the TypeLibConverter class in the System.Runtime.InteropServices namespace.
- The function returns an AssemblyBuilder object which contains the server's events, which we can reflect on during runtime.
- This AssemblyBuilder object is nothing but a wrapper created at runtime.
- This is similar to the wrapper we create using TlbImp.exe SDK.

# Adding Event Handlers at runtime

- Reflect on the AssemblyBuilder object at runtime and get the events exposed by the server.
- Create a delegate of the eventhandler type associated with this event at runtime using System.CreateDelegate method.
- This delegate encapsulates the method to be invoked when the event is raised by the server.
- Add this EventHandler to the COM object we created using Activator.CreateInstance.

# References


- .Net and COM – The Complete Interoperability Guide, Adam Nathan, SAMS, 2002