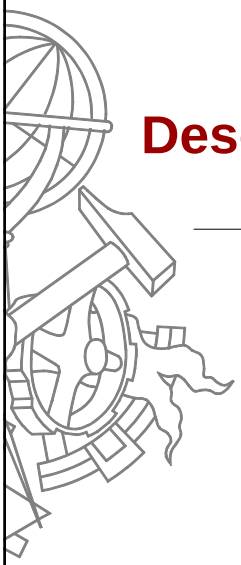


Ambientes de Desenvolvimento Avançados

http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

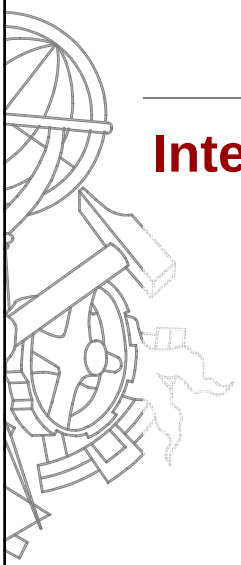
Aula 15

Engenharia Informática

2006/2007

José António Tavares
jrt@isep.ipp.pt

1



Interoperabilidade

Conteúdo

- Introduction to interoperation between
  - .NET and COM
  - COM and .NET
  - .NET and platform services

Existing Stuff Must Coexist

- Transition to .NET will be a lasting process
- Portions of systems will remain as they are now
  - ...and even maintained in their current form
- Interoperability increases the momentum of .NET
- Goal is to achieve total transparency
- Mere presence of .NET must not interfere with COM
- Performance should not be hit

Differences in the two Worlds?

- Managed Code vs. Unmanaged Code
  - Lifetime management
  - Metadata
- Common Type System
- Inheritance concept
- Threading model
- Error handling mechanisms
- Registration

COM and platform interoperation

- CLR includes substantial support for COM interoperation and direct access to the underlying platform
- By integrating support for both COM interoperation and platform invocation, the CLR execution engine can deliver almost optimal performance.
- COM interoperation is achieved by providing two kinds of automatically synthesized wrappers
  - COM-callable wrappers present CLR objects via COM interfaces;
  - Runtime-callable wrappers present COM objects via CLR interfaces.

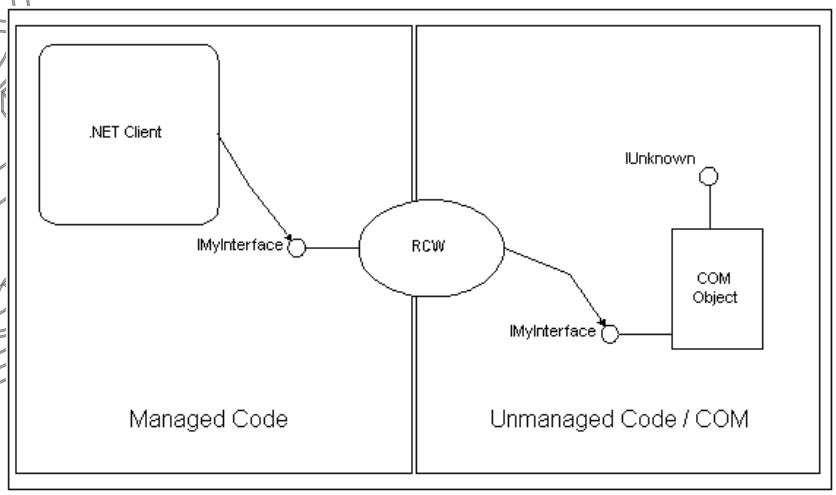
General .NET Interoperability

- .NET components and clients are managed code running under the CLR (common language runtime).
- COM (Component Object Model) components are not managed and are controlled by the common language runtime.
- To bridge the gap between the managed client and unmanaged server (COM), we must use the Runtime Callable Wrapper (**RCW**).

Understanding the Interaction

- The .NET client consuming the COM component must call the RCW.
- The RCW is generated as a DLL that lies between your managed code and the COM component that is being consumed.
- The COM component now operates as if it were called from a COM consumer.

The Runtime Callable Wrapper (RCW)



The RCW Handles

- **Object binding**
  Both early and late bound interfaces are supported.
- **Data marshaling and translation**
  Data type conversion is handled between managed and unmanaged data types.
- **Object lifetime management**
  Object references are managed to ensure that objects are either released or marked for garbage collection.
- **Object identity**
  COM object identity rules are enforced.
- **Exception and error handling**
  The runtime translates COM HRESULT values to .NET exceptions and vice versa.

Where Do We Get the RCW?

- The RCW can be created in a number of ways, but Microsoft Visual Studio® .NET will create it for you when references are set to a COM component.
- RCW can also be created using the utility `Tlbimp.exe` that is shipped with the Microsoft .NET Framework SDK.

Note: The RCW, when created, needs to be added to the application's Bin directory.

Calling COM Services from .NET

- Metadata for COM class must be available
  - Can be generated from Type Library
  - Can declare Custom Wrapper
- Use COM class like .NET class
- Early and late bound activation
- COM component must be registered locally

Converting Type Library to Metadata

- Use SDK tool `TlbImp`
  - Or just add reference to server in Visual Studio
- Type library can be imported from executable
- Reference resulting metadata file in project
- Custom IDL attributes are preserved

Creating the RCW

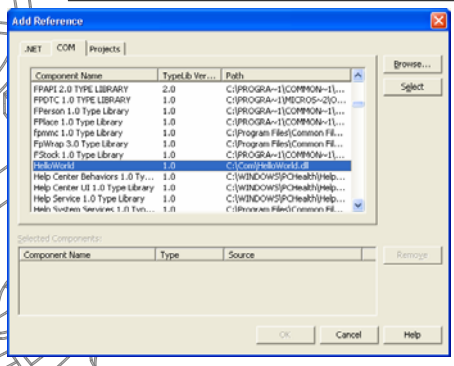
- To work with the COM object, we must use `Tlbimp.exe` to create the RCW.
- When we create the RCW, we may need to pass the parameter `/out:RCW_`*`FileName`*`.dll` to specify the name of the RCW DLL. (Where RCW_*FileName*.dll is the file name of the RCW we make up.)
  - `Tlbimp HelloWorld.dll /out:RCW_HelloWorld.dll`

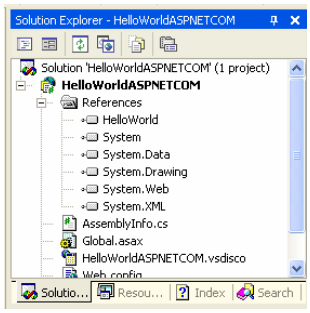
```
C:\Com>tlbimp HelloWorld.dll /out:RCW_HelloWorld.dll
Microsoft (R) .NET Framework Type Library to Assembly Converter 1.0.3705.0
Copyright (C) Microsoft Corporation 1998-2001.  All rights reserved.

Type library imported to C:\Com\RCW_HelloWorld.dll

C:\Com>_
```

Creating the RCW in Visual Studio .NET





There are at least two ways to create the RCW. One way is to just add the reference in Visual Studio .NET. Visual Studio .NET will create the RCW and add it to the application's Bin directory.

Calling an COM Server

```
namespace CallingCOMServer
{
   using System;
   using COMServerLib;

   public class DotNET_COMClient
   {...
        public static int Main(string[] args)
        {
              MyCOMServer myS = new MyCOMServer();
              return (myS.Add(17,4));
        }
   }
};
```

Late Bound Activation

- Accomplished by Reflection API
  - No difference whether COM or .NET object
- Type can be obtained from **`ProgID`** or **`ClsID`**
- **`InvokeMember`** to access methods and properties
- Metadata is not needed, but useful
  - COM object is wrapped by `__ComObject`

Late Bound Call to COM Server

```
namespace LateBoundClient
{
   using System.Reflection;
   ...
   Type typ;
   Object obj;
   Object[] prms = new Object[2];
   int r;
   typ = Type.GetTypeFromProgID("MyLib.MyServer");
   obj = Activator.CreateInstance(typ);
   prms[0] = 10;
   prms[1] = 150;
   r = (int)typ.InvokeMember("aMethod",
        BindingFlags.InvokeMethod, null, obj, prms);
   ...
}
```

Calling .NET Services from COM

- Use .NET class like COM class
  - Type and methods must be public
- Early and late bound activation
- Convert Metadata to Type Library
- Wrapper of .NET component must be registered
  - Can use **`RegAsm`** tool

Strong-Named Assembly

- Global Assembly Cache
  - Contains assemblies shared by unrelated applications
- Strong Names Are Required for Assemblies in the Cache
  - Generate a public-private key pair:

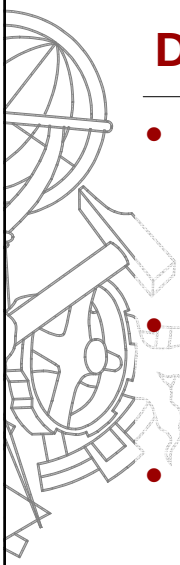
```
sn –k orgKey.snk
```

  - Add code to source file to specify version and key information:

```
#if STRONG
[assembly: System.Reflection.AssemblyVersion("1.0.0.0")]
[assembly: System.Reflection.AssemblyKeyFile("orgKey.snk")]
#endif
```

  - Compile:

```
csc /define:STRONG /target:library /out:AReverser.dll➥
 AReverser.cs
```



Deploying Shared Components

- Installing the Strong-Named Component in the Global Assembly Cache

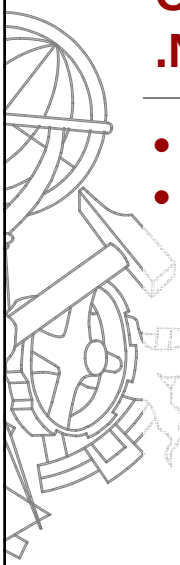
```
gacutil /i AReverser.dll
```

- Examining the Global Assembly Cache

```
gacutil /l
```

- Removing a Shared Component File

```
gacutil /u AReverser
```

Calling Platform Services from .NET

- Calling static functions in DLLs
- P/Invoke provides services
  - Locates implementing DLL
  - Loads DLL
  - Finds function address
    - Fuzzy name matching algorithm
  - Pushes arguments on stack
  - Performs marshalling
  - Enables pre-emptive garbage collection
  - Transfers control to unmanaged code

P/Invoke Sample

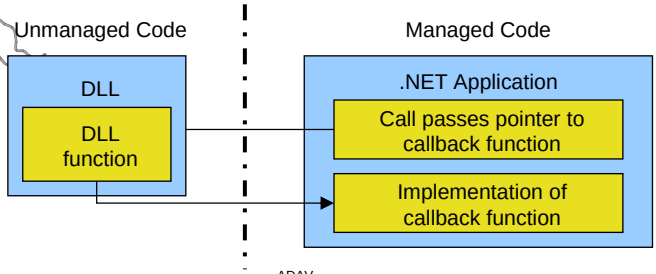
```
namespace HelloWorld
{
  using System;

  class MyClass
  {
    [dllimport("user32.dll", CharSet=CharSet.Ansi)]
    static extern int MessageBox(int h, string m,
                                 string c, int t);

    public static int Main()
    {
      return MessageBox(0, "Hello World!", "Caption", 0);
    }
  }
}
```


Introduction

- Unmanaged code can call back to managed code
  - Unmanaged parameter is function pointer
  - Must supply parameter as delegate
  - P/Invoke creates callback thunk
    - Passes address of thunk as callback parameter



Callback Sample

```
public class EnumReport
{
  public bool Report(int hwnd, int lParam)
  { // report the window handle
    Console.Write("Window handle is ");
    Console.WriteLine(hwnd);
    return true;
  }
};
public class SampleClass
{
  delegate bool CallBack(int hwnd, int lParam);

  [DllImport("user32")]
  static extern int EnumWindows(CallBack x, int y);
  public static void Main() {
    EnumReport er = new EnumReport();
    CallBack myCallBack = new CallBack(er.Report);
    EnumWindows(myCallBack, 0);
  }
}
```


Questões

?