German Jordanian University الجامعة الألمانية الأردنية

CS355 Web Technologies

Dr. Ismail Hababeh

German-Jordanian University

Lecture 23

.NET Distributed Services - Remote Connectors

- A service device or a client cannot directly access a remote service (.EXE) running in different application domains in the same or different processors.
- Access can be done using Remote Channel Connection.
- The marshaling makes it possible to invoke a remote method of a distributed service.

.NET Distributed (Remote) Services Marshaling

There are two ways to invoke a remote method (marshal an object):

- MBV (Marshal by Value) server passes a copy of an object to a client.
- MBR (Marshal by Reference) client creates a proxy of a remote object. MBR should be used when a remote service runs at a remote site.

Remote Channel Connection RCC

- RCC is a way to implement communications between clients and remote services. (e.g TCP)
- RCC is like the socket communication in Java.
- Each RCC must be bind (registered) with a port:
  - Client channel binds a client port
  - Server channel binds a server port

Remote Channel Connection RCC

- Example:
  - The client needs to get an object of the remote service Temperature Convertor.
  - The Temperature Convertor service is on a remote server at port 4000.
  - Create TCP channel on server port 4000 and register it with the remote class and URL name (Server channel).
  - Create a TCP channel and register it on the client port (Client channel).

Remote Connectors for .NET Distributed Services - Example

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

public class TempConv  MarshalByRefObject
{
   public static void Main()
    {
     TcpChannel channel = new TcpChannel(4000);   // create server channel
     ChannelServices.RegisterChannel(channel);
     RemotingConfiguration.RegisterWellKnownServiceType (
            typeof(TempConv),
            "TempConvDotNet",
            WellKnownObjectMode.Singleton );
     System.Console.WriteLine("Press <enter> to exit...");
     System.Console.ReadLine();
   }
```

```
public double cToF(double c)
   {
      return (int)(c*9/5.0+32);
   }

public double fToC(double f)
   {
      return (int)(f-32)*5/9.0;
   }
}
```

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
class MainApp
{
    public static void Main()
    {
```

```
try          // create client channel
{
TcpChannel channel = new TcpChannel();
ChannelServices.RegisterChannel(channel);
TempConv  myTempConv =
        (TempConv)Activator.GetObject(
                typeof(TempConv),
                "tcp://127.0.0.1:4000/TempConvDotNet");
          double choice;
          double input;
          double output;
          boolean next = true;
```

```
while (next)
    {
Console.WriteLine("Please enter your choice:
    1. Convert from F to C, 2. Convert from C to F, 3. Exit");
          choice = Integer.Parse (Console.ReadLine());
           if (choice == 1)
            {
                  Console.WriteLine("Input temperature in F: ");
                  input=Double.Parse(Console.ReadLine());
                  output = myTempConv.fToC(input);
                  Console.WriteLine(output);
            }
```

```
else if (choice ==2)
    {
        Console.WriteLine("Input temperature in C: ");
        input=Double.Parse(Console.ReadLine());
        output = myTempConv.cToF(input);
        Console.WriteLine(output);
    }
else
    {
        next= false;
        Console.WriteLine ("Thank you for using .Net");
    }
   }
  }
```

```
catch (Exception e)
    {
        Console.WriteLine(e.ToString());
    }
  }
}
```

Remote Connectors for .NET Distributed Services

- Steps to build the server and client services:

```
>csc TempConv.cs
>csc /r:TempConv.exe TempConvClient.cs
```

Activate the distributed server component and its client:

```
>Tempconv.exe
>TempConvClient.exe
```

.NET Communication Model - Remote Asynchronous Callback

- The Remote asynchronous callback is based on Remote Delegate. The client is not blocked while waiting for notification from remote services.
- Example: Client wants to be notified once the stock prices reaches a specified level. Instead of pooling the stock price all the time, let the server notify the client when the job is done.
- When client makes asynchronous call to a remote method of remote service, it passes a callback method to the server to be called back late through Remote Delegate.

Remote Asynchronous Callback Between Distributed .NET Services

- Two Delegates will occur:
  - Mydelegate pointing to the remote method "cToF" of remote service named "TempConvDotNet".
  - AsynchCallback pointing to callback method "MyCallBack".

Example:

```
using System;
using System.Threading;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Messaging;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
```

Remote Asynchronous Callback Between Distributed .NET Services - Example

```
public class Client {
    public delegate double MyDelegate(double c)
public static int main(string [] agrs)
TcpChannel chan =  new TcpChannel();  // step1: create a channel
ChannelServices.RegisterChannel(chan); // step2: register the channel
TempConv obj = (TempConv)Activator.GetObject(typeof(TempConv),
    "tcp://localhost:4000/TempConvDotNet"); // step3: get the object
If {(obj ==  null) System.Console.WriteLine("Couldn't locate server");}
else { // step1: create a Asynchronous Callback
    AsyncCallback cb = new AsyncCallback(Client.MyCallBack);
    MyDelegate d = new MyDelegate(obj.cToF); //create a delegate
    // that points to the remote method
    AsyncResult ar = d.BeginInvoke(32, cb, null); // invoke callback
}
```

```
System.Console.WriteLine("Press <enter> to exit … ");
System.console.ReadLine();
return 0;
}
public static void MyCallBack(AsyncResult ar)
{
    MyDelegate d = (MyDelegate)((AsyncResult)ar).AsyncDelegate;
    Console.WriteLine(d.EndInvoke(ar));
    Console.WriteLine("Done..");
}
}
```

Notes on the previous example

- The first parameter of BeginInvoke method is a 32 degrees in Celsius to be converted in the “cToF” remote method.
- The second parameter cb is a callback Delegate to prevent blocking the client program.
- When the remote method completes the conversion work, the callback method is called and AsyncResult is returned to client.
- The third parameter is an object used to pass a state. (in this case null). The callback function is invoked when setState finished and the component gets rendered.