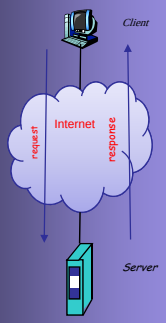


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 9.1: Remoting (Part 1)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

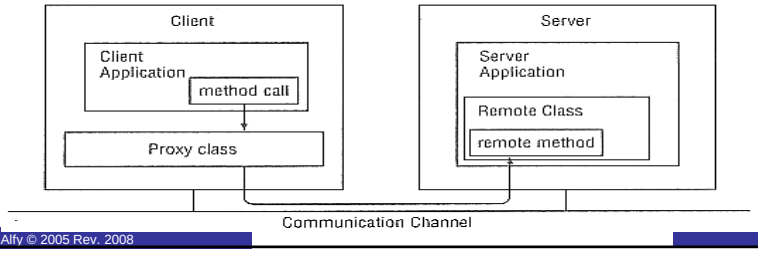
- Learn the basics of remoting
- Learn the .NET remoting architecture
- Learn the components of a remoting application and how to write each of them.
- Learn how to configure remoting server and remoting client declaratively.

What is Remoting?

- Remoting is a distributed system in which an application communicates with another application running in a different application domain by invoking its methods.
  - The two application domains can be on the same computer or on different computers attached to the same of different networks.
- Remoting provides another way of writing C/S applications where, instead of exchanging messages, a client gets a service by directly calling a remote method.
- This approach allows C/S applications to be written at a higher level, very much like writing stand-alone application, relieving the programmer from:
  - The need to create and abide by message exchange protocols
  - The need to convert data back-and-forth from binary to its real type.

.NET Remoting Architecture

- Methods that will be called from the client are implemented in a remotable class.
- Instance(s) of the remotable class (the remote objects) are hosted by a remoting server.
- When a client creates an instance of the remotable class, the .NET system creates a proxy object on the client side.
  - The proxy object acts like the real object to the client with the same public methods.
  - When the client calls a method of the proxy object, the proxy object calls the corresponding method of the remote object



.NET Remoting Architecture …

- Communication between the client (through the proxy object) and the remote object takes place through a Communication Channel.
- All *objects exchanged* between the client and the remote object are automatically serialized and Deserialized by the system using formatter classes.
  - Objects exchanged are parameters passed as arguments (by the client) to remote methods or values returned by the remote methods.
  - Serialization is the process of converting an object into a stream so that it can be saved on disk or transmitted over the network. The process of recovering the object from stream is called Deserialization.

Formatters and Channels

- .NET provides two formatter classes that automatically serialize objects exchanged in a remoting applications:
  - System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
  - System.Runtime.Serialization.Formatters.Soap.SoapFormatter
  - BinaryFormatter serializes an object into a stream of bytes.
  - SoapFormatter serializes an object into XLM using the Simple Object Access Protocol (SOAP) - the standard for exhanging XML data.
  - BinaryFormatter is more compact and therefore more efficient.
- .NET also provides two types of communication channels:
  - System.Runtime.Remoting.Channels.Tcp.TcpChannel
  - System.Runtime.Remoting.Channels.Http.HttpChannel.
  - TcpChanned is the default for exchanging objects serialized using BinaryFormatter. It may be restricted to LAN by firewalls.
  - HttpChannel is used to exchange messages over the internet and is the default for exchanging data formatted using SoapFormatter.
  - The best approach is to use BinaryFormatter over HttpChannel

Writing a Serializable class

- All standard data type classes (Int32, Double, String, etc.) are serializable and thus, can be used in a remotable class.
- However, user-defined classes must be declared as serializable by adding the *Serializable* attribute at the top:

```
1.  using System;

2.  [Serializable]
3.  public class Person {
4.      private int age;
5.      private string name;
6.      public Person(int age, string name) {
7.        this.age = age;
8.        this.name = name;
9.      }
10.     public override string ToString() {
11.       return "Name: " + name + ", Age: " + age;
12.     }
13. }
```


Writing a Serializable class …

```
1.  using System;
2.  using System.IO;
3.  using System.Runtime.Serialization.Formatters.Binary;
4.  public class SerialTest {
5.      public static void Serialize() {
6.          Person p = new Person(25, "Amir");
7.          FileStream fs = new FileStream("temp.dat", FileMode.Create);
8.          BinaryFormatter formatter=new BinaryFormatter();
9.          formatter.Serialize(fs,p);  fs.Close();
10.     }
11.     public static void DeSerialize() {
12.         FileStream fs= new FileStream("temp.dat", FileMode.Open);
13.         BinaryFormatter formatter=new BinaryFormatter();
14.         Person recoveredPerson =(Person) formatter.Deserialize(fs);
15.         Console.WriteLine(recoveredPerson);   fs.Close();
16.     }
17.     public static void Main(string[] s) {
18.         Serialize();
19.         DeSerialize();
20.     }
21. }
```

```
Name: Amir, Age: 25
Press any key to continue . . .
```

Writing a Remoting Application

- Writing a remoting application involves writing the following elements:
  - A remotable class
  - A remoting server
  - A remoting client

Remotable class

- This is a class whose methods can be accessed remotely from another application domain.
- To construct a remotable class, all that is needed is to extend the **System.MarshalByRefObject** class.

Writing a Remoting Application …

```
1.  using System;
2.  public class MathClass:MarshalByRefObject {
3.    public double Add(double a, double b) {
4.      return a + b;
5.    }
6.    public double Subtract(double a, double b) {
7.      return a - b;
8.    }
9.    public double Multiply(double a, double b) {
10.     return a * b;
11.   }
12.   public double Divide(double a, double b) {
13.     if (b == 0)
14.       return 0;
15.     else
16.       return a/b;
17.   }
18. }
```

This class needs to be compiled into a a DLL

Writing a Remoting Application …

Remoting Server

- This is an application that hosts the remote object.
  - A client connects to a server in order to gain access to the methods of the remote object.
- Writing a Remoting server involves three steps:

1. Create a TCP or HTTP communication channel using either **TcpChannel** or **HttpChannel** respectively.

```
HttpChannel channel = new HttpChannel(9095);
```

2. Register the communication channel with the remoting channel services. This is done using the static method, ***RegisterChannel***, of the, **System.Runtime.Remoting.Channels.ChannelServices** class.

```
ChannelServices.RegisterChannel(channel);
```

3. Register the remotable class with the remoting server. This is done using the static method, ***RegisterWellKnownServiceType,*** of the **System.Runtime.Remoting.RemotingConfiguration** class.

Writing a Remoting Application …

- The ***RegisterWellKnownServiceType*** method takes three arguments:
  - The *type* of the remotable class
  - A *URI* identifier for the class, and
  - Object creation *mode*.
    - Possible modes are: **SingleCall** and **Singleton**, both of which are fields of the **WellKnownObjectMode** class.
    - **SingleCall** means a separate instance of the remotable class will be created for each method call to the remotable class - useful if you do not need to maintain object state across method calls.
    - **Singleton** mode means, a single instance will be used for different calls for all clients. Singleton is useful if you wish to retain the state across different calls.

Writing a Remoting Application …

```
1.  using System;
2.  using System.Runtime.Remoting;
3.  using System.Runtime.Remoting.Channels;
4.  using System.Runtime.Remoting.Channels.Http;

5.  public class MathServer {
6.    public static void Main() {
7.      HttpChannel channel = new HttpChannel(9095);
8.      ChannelServices.RegisterChannel(channel);
9.      RemotingConfiguration.RegisterWellKnownServiceType(
10.           typeof(MathClass),  //type of Remotable class
11.           "MyMathServer",     //user-defined object URI
12.            WellKnownObjectMode.SingleCall); //Mode
13.     Console.WriteLine("Press <enter> to exit...");
14.     Console.ReadLine();
15.   }
16. }
```

The **MathClass.dll** should be stored in the same folder as the MathServer.exe

Writing a Remoting Application …

Remoting Client

- This is the application that accesses the methods of the remotable class through the remoting server.
- Writing a remoting client involves three steps:

1. Create a Channel using either **TcpChannel** **or** **HttpChannel**
   - The channel must be of same type as that of the remoting server.
2. Register the channel with the remoting channel services.
3. Creating an instance of the proxy class.

Writing a Remoting Application …

- The Proxy object can be created using the method, ***RegisterWellKnownClientType*** of the **RemotingConfiguration** class.
  - The method takes the type of the remote object and its URI as arguments.

  Example:

```
RemotingConfiguration.RegisterWellKnownClientType(
    typeof(MathClass),  //type of Remotable object
    "http://localhost:9095/MyMathServer");  //URI
MathClass math = new MathClass();
```

- Alternatively, the ***getObject*** method of the **System.Activator** class is used.

  Example:

```
MathClass math = (MathClass)Activator.GetObject(
      typeof(MathClass),
      "http://localhost:9095/MyMathServer");
```


Writing a Remoting Application …

- Whatever method is used in creating the proxy object, the type the of remote object must exist on the client side.
  - Cannot declare a reference variable without a type!
- Two options are possible:

1. Provide the assembly containing the remotable class to the client.
   - Question: Why then do we need the remotable object if we have its class locally at the client side?
   - Answer: The remote object usually provides some service that a local instance cannot provide.
2. Provide an interface containing the methods of the remotable class to the client.
   - The remotable class must be designed to implement the interface.

Writing a Remoting Application …

```
1.  using System;
2.  using System.Runtime.Remoting;
3.  using System.Runtime.Remoting.Channels;
4.  using System.Runtime.Remoting.Channels.Http;
5.  public class MathClient {
6.     public static void Main(string[] args) {
7.       HttpChannel channel = new HttpChannel();
8.       ChannelServices.RegisterChannel(channel);
9.       RemotingConfiguration.RegisterWellKnownClientType (
10.          typeof (MathClass), "http://localhost:9095/MyMathServer" );
11.      MathClass math = new MathClass();
12.      if (math == null)
13.        Console.WriteLine("Could not locate Server");
14.      else   {
15.        int a = 10, b = 5;
16.        Console.WriteLine("{0} + {1} = {2}", a, b, math.Add(a, b));
17.        Console.WriteLine("{0} - {1} = {2}", a, b, math.Subtract(a, b));
18.        Console.WriteLine("{0} * {1} = {2}", a, b, math.Multiply(a, b));
19.        Console.WriteLine("{0} / {1} = {2}", a, b, math.Divide(a, b));
20.      }
21.   }
22. }
```


Declarative Configuration

- The last example shows the **programmatic configuration** of the remoting server and the remoting client.
  - The configuration information is specified inside the source code.
  - The disadvantage of programmatic configuration is that if the server is moved to another machine, then the source code must be modified and recompiled.
- An alternative approach involves specifying the configuration information using XML tags in a text file.
  - The server and the clients are then written so that they read the information from the text files.
  - If there is any change in the information, the text files can be easily modified accordingly.
  - This type of configuration is called **declarative configuration**.

Declarative Configuration …

//File: MathServer.config

```
<configuration>
 <system.runtime.remoting>
  <application>
   <channels>
    <channel ref="http" port="9090">
     <serverProviders>
           <provider ref="wsdl" />
      <formatter ref="soap" typeFilterLevel="Full" />
      <formatter ref="binary" typeFilterLevel="Full" />
     </serverProviders>
    </channel>
   </channels>
   <service>
       <wellknown mode="Singleton" type="MathClass, MathLibrary"
          objectUri="MyMathServer" />
   </service>
  </application>
 </system.runtime.remoting>
</configuration>
```


Declarative Configuration …

Notes:

- If MathClass is in a namespace, Math, then you must include it in the *type* specification as:

  type="Math.MathClass, MathLibrary"

  – This is important since Visual studio automatically encloses classes in a namespace.
- In the example, both soap and binary formatters are specified to be used over a http channel.
- The configuration is fairly general. All that is needed to adapt it to another application is to modify the service part to reflect the remotable class and possibly change the port number.

Declarative Configuration …

- With the configuration file, the MathServer program reduces to few lines as shown below:

```
1.  using System;
2.  using System.Runtime.Remoting;

3.  public class MathServer
4.  {
5.    public static void Main() {
6.      RemotingConfiguration.Configure("MathServer2.config");
7.      Console.WriteLine(
8.          "Server started, press Enter to terminate...");
9.      Console.ReadLine();
10.   }
11. }
```

Only the configuration file needs to be changed for the server to work with another remotable object.

- Note: Since there is no reference to the MathClass in the code, the DLL file containing the class must be copied manually to the location of the of the server.

Declarative Configuration …

//File: MathClient2.config

```
<configuration>
 <system.runtime.remoting>
  <application>
   <channels>
    <channel ref="http" port="0">
     <clientProviders>
      <formatter ref="binary" />
     </clientProviders>
    </channel>
   </channels>
   <client>
        <wellknown type="MathClass, MathLibrary"
          url="http://localhost:9090/MyMathServer" />
   </client>
  </application>
 </system.runtime.remoting>
</configuration>
```

Declarative Configuration …

```
1.  using System;
2.  using System.Runtime.Remoting;
3.  public class MathClient {
4.    public static void Main(string[] args)
5.    {
6.      RemotingConfiguration.Configure("MathClient2.config");
7.      MathClass math = new MathClass();
8.
9.      if (math == null)
10.       Console.WriteLine("Could not locate Server");
11.     else
12.     {
13.       int a = 10, b = 5;
14.       Console.WriteLine("{0} + {1} = {2}", a, b, math.Add(a, b));
15.       Console.WriteLine("{0} - {1} = {2}", a, b, math.Subtract(a, b));
16.       Console.WriteLine("{0} * {1} = {2}", a, b, math.Multiply(a, b));
17.       Console.WriteLine("{0} / {1} = {2}", a, b, math.Divide(a, b));
18.       Console.ReadLine();
19.     }
20.   }
21. }
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm