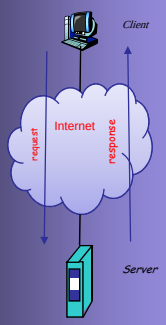

INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 9.2: Remoting (Part 2)

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

Objectives

- Learn about types of Remote Objects
  - Server Activated Objects (SAO)
  - Client Activated Objects (CAO)
- Learn about the life-time of remote objects
- Learn how to create event-generating remote objects.

Server -Activated Objects (SAO)

- A remote object is said to be **Server-Activated** (SAO), if when a client creates an instance of the remotable class, the instance is not created on the server until a method is called.
- Both **SingleCall** and **Singleton** modes of creating objects are SAO.
- When the client creates a Proxy object, the real object is not created on the server side until a method is called.

Client-Activated Objects (CAO)

- In **client-activated**, an object is created on the server side the moment the client uses *new* to create the proxy object.
- Client-activated object provides two advantages:
  - It can be used to call a non-default constructor of the remote class. This is not possible in server-activated modes.
  - It provides another type of life-time, different from those of **SingleCall** and **Singleton**. A separate object is created for each client and the state of the object is maintained across multiple calls.
- Use Client-Activated if object state needs to be maintained separately for each client.

Client-Activated Objects (CAO) …

- An example application best implemented using CAO is a remote Stop-Watch service.
- The state needs to be maintained (from start time to stop time) and each client needs its own instance.

```
1.  using System;

2.  public class StopWatch : MarshalByRefObject {
3.      DateTime start = DateTime.Now;

4.      public void Start () {
5.          start = DateTime.Now;
6.      }
7.      public int Stop () {
8.          return (int)((DateTime.Now - start).TotalMilliseconds);
9.      }
10. }
```


Client-Activated Objects (CAO) …

- Client and Server configurations for CAO are different from those of SAO.

//StopWatchServer.config

```
<configuration>
   <system.runtime.remoting>
      <application>
         <channels>
            <channel ref="http"  port="8080" />
         </channels>
         <service>
            <activated type="StopWatch, StopWatchLibrary"/>
         </service>
      </application>
   </system.runtime.remoting>
</configuration>
```

- Activated service is used instead of wellknown
- No need for URI and Mode paras.
- Formatters are declared on the client side

Client-Activated Objects (CAO) …

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

3.  class StopWatchClient {
4.      static void Main () {
5.          RemotingConfiguration.Configure(
6.                                  "StopWatchClient.config");
7.          StopWatch sw = new StopWatch ();
8.          sw.Start ();

9.          Console.WriteLine(
10.                     "Press Enter to show elapsed time …");
11.         Console.ReadLine ();
12.         Console.WriteLine (sw.Stop () + " millseconds");
13.          Console.ReadLine();
14.     }
15. }
```


Client-Activated Objects (CAO) …

//StopWatchClient.config

```
<configuration>
   <system.runtime.remoting>
      <application>
         <client url="http://localhost:8080">
            <activated type="StopWatch, StopWatchLibrary"/>
         </client>
         <channels>
            <channel ref="http" port="0">
               <serverProviders>
                <formatter ref="soap" typeFilterLevel="Full"/>
                <formatter ref="binary" typeFilterLevel="Full"/>
               </serverProviders>
            </channel>
         </channels>
      </application>
   </system.runtime.remoting>
</configuration>
```

Life-Time of Remote Objects

- Life-time of a remote object depends on its activation mode:
  - For **SingleCall** objects**,** it is the duration of the method call
  - For **Singleton** and **Client-Activated** objects, it is controlled by a lease.
- A lease is an object that implements the **ILease** interface of the **System.Runtime.Remoting.Lifetime** namespace.
- The ILease interface has the following properties:

TimeSpan InitialLeaseTime	Length of time following activation that the object lives if it receives no method calls
TimeSpan RenewOnCallTime	Minimum value that the CurrentLeaseTime is set to each time the object receives a call
TimeSpan CurrentLeaseTime	Amount of time remaining before the object is deactivated if it does not receive a method call

Life-Time of Remote Objects …

LeaseState CurrentState	Gets the current LeaseState of the lease. **LeaseState** is an enumeration with values: Active, Expired, Initial, Null, Renewing

- The default for **InitialLeaseTime** is 5 minutes, and the default for **RenewOnCallTime** is 2 minutes.
  - The defaults can be changed by overriding ***InitializeLifetimeService*** method of the base **MarshalByRefObject** class.

```
1.   public class RemotableClass : MarshalByRefObject {
2.      public override object InitializeLifetimeService ()     {
3.          ILease lease = (ILease) base.InitializeLifetimeService ();
4.          if (lease.CurrentState == LeaseState.Initial)  {
5.              lease.InitialLeaseTime = TimeSpan.FromMinutes (20);
6.              lease.RenewOnCallTime = TimeSpan.FromMinutes (10);
7.          }
8.          return lease;
9.      }
10.  // ...
11. }
```

Life-Time of Remote Objects …

- To make an object to remain active forever, override the *InitializeLifeTimeServices* method to return null:

```
1.    public class RemotableClass : MarshalByRefObject {
2.       public override object InitializeLifetimeService ()     {
3.             return null;
4.       }
5.    // ...
6.   }
```

- The default times can also be changed declaratively:

```
 <configuration>
  <system.runtime.remoting>
    <application>
      <lifetime leaseTime="20M" renewOnCallTime="10M" />
    </application>
  </system.runtime.remoting>
</configuration>
```

Other suffixes for the time span are : D for days, H for hours or S for seconds - the default.

Event-Generating remote objects

- In addition to providing remote methods, remote objects can also provide remote events.
  - A client register for a remote event using its local method.
  - When the event occurs, the remote object notifies the client and the local method get executed.
  - In this way, it is easy to create a broadcasting system using remoting.
  - All clients register for the event with the remote object. Each time the event occurs, all registered clients are notified.

Event-Generating remote objects …

- To create remotable class that can generate events, the following additional settings are required:
  - The .exe of the client must be copied on the server side. Remote object needs this to bind its event with the method that the client used to register for the event.
  - To make the system efficient, the method that is invoked by the remote object to notify its clients when the event occurs should be tagged with the **[OneWay]** attribute.
  - [OneWay] is a shorthand for *OneWayAttribute*, which is in the **System.Runtime.Remoting.Messaging** namespace.
  - Calls to methods tagged as OnWay runs **asynchronously**. Also the caller will not be notified of the result of the call.
  - Such methods are called **fire-and-forget**.

Event-Generating remote objects …

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

4.  public delegate void MessageHandler(string msg);

5.  public class ChatClass : MarshalByRefObject {
6.     public event MessageHandler MessageSender;
7.
8.      public override object InitializeLifetimeService () {
9.          return null;
10.     }

11.    [OneWay]
12.    public void SendMessage(string msg) {
13.      if (MessageSender != null)
14.        MessageSender(msg);
15.    }
16. }
```

Event-Generating remote objects …

```
1.  using System;
2.  using System.Windows.Forms;
3.  using System.Runtime.Remoting;
4.  using System.ComponentModel;

5.  public class ChatClient : System.Windows.Forms.Form
6.  {
7.       private System.Windows.Forms.TextBox inBox;
8.       private System.Windows.Forms.Button sendBt;
9.       private System.Windows.Forms.GroupBox groupBox;
10.      private System.Windows.Forms.GroupBox groupBox2;
11.      private System.Windows.Forms.TextBox outBox;
12.
13.      private MessageHandler handler;
14.      private ChatClass chatObject;
```


Event-Generating remote objects …

```
15.     public ChatClient()  {
16.       InitializeComponent();
17.       try {
18.          RemotingConfiguration.Configure ("ChatClient.config");
19.          chatObject = new ChatClass();
20.          handler = new MessageHandler(OnNewMessage);
21.          chatObject.MessageSender +=  handler;
22.       }
23.       catch (Exception ex) {
24.          MessageBox.Show (ex.Message);
25.          Close ();
26.       }
27.     }
28.     void InitializeComponent() {
29.       //deleted
30.     }
31.     public static void Main(string[] args) {
32.        Application.Run(new ChatClient());
33.     }
```

Event-Generating remote objects …

```
34. public void OnSendClicked(object sender, System.EventArgs e)
35.   {
36.       chatObject.SendMessage(outBox.Text);
37.   }
38.
39.   public void OnNewMessage(string msg) {
40.       inBox.Text += msg + "";
41.   }

42.   protected override void OnClosing (CancelEventArgs e)
43.   {
44.      // Disconnect event handler before closing
45.       base.OnClosing (e);
46.       chatObject.MessageSender -= handler;
47.   }
48. }
```


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