.NET Deployment

Matt Smouse

CSE775 – Distributed Objects

Spring 2003

Outline

- Deployment issues
- Configuration files
- Soapsuds and implementation hiding
- Server Deployment with Windows Services
- Server Deployment with IIS
- Client Deployment with IIS

Deployment Issues

- Change in server location
  - Does the client hard-code the location and port of remote objects on the server?
- Uses of the application
  - Will this application be used in other ways? For instance, LAN vs Internet use.
- New/additional remotable objects
  - Will we be adding remotable objects after we have built the application?
- Web deployment
- Implementation hiding
  - Do we want to allow the client to disassemble our code?

Configuration Files

- Rather than hard-code the registration of remote objects and their channels, we can use a configuration file.
- Using a configuration file allows us to do the following without recompiling the server or client:
  - Change the type of channel that is used
  - Add additional remotable objects
  - Change the lifetime settings of remotable objects
  - Add message sinks or formatters to the server or client
- This functionality is available through the System.Runtime.Remoting assembly.

Configuration Files (cont)

- A configuration file is an XML document that is loaded by the server or client.
- Use two different configuration files for the client and the server.
- On the server, load the configuration file using RemotingConfiguration.Configure(“MyServer.exe.config”);
- On the client, load the configuration file using RemotingConfiguration.Configure(“MyClient.exe.config”);
- After loading the configuration file on the client, simply call new on the remotable object class to create a proxy.

Configuration Files (cont)

- Content and structure

```
<configuration>
    <system.runtime.remoting>
        <application>
            <lifetime />
            <channels />
            <service />
            <client />
        </application>
    </system.runtime.remoting>
</configuration>
```

Configuration Files (cont)

- Lifetime
  - The <lifetime> tag allows you to change the lifetime of your remotable objects.
  - Valid attributes:
    - leaseTime – This is the initial lease time that an object will have to live before it is destroyed.
    - sponsorshipTimeout – The time to wait for a sponsor’s reply.
    - renewOnCallTime – This is the additional lease time that is added with each call on the remote object.
    - leaseManagerPollTime – Specifies when the object’s current lease time will be checked.
  - Note that these apply to Singleton and Client-Activated objects only.

Configuration Files (cont)

- Channels
  - The <channels> element contains the channels that your application will be using. We declare channels with the <channel> tag.
  - The <channel> tag specifies the type, port, and other properties for a particular channel.
  - Valid attributes:
    - ref – “http” or “tcp”
    - displayName – Used for .NET Framework Configuration Tool
    - type – if ref is not specified, contains namespace, classname, and assembly of the channel implementation.
    - port – server side port number. Use 0 on the client if you want to get callbacks from the server.
    - name – Unique names to specify multiple channels (use “”)
    - priority – Sets priority of using one channel over another.

Configuration Files (cont)

- Channels
  - Valid attributes (cont):
    - clientConnectionLimit – Number of simultaneous connections to a particular server (default = 2)
    - proxyName – name of the proxy server
    - proxyPort – port of the proxy server
    - suppressChannelData – specifies whether a channel will add to the ChannelData that is sent when an object reference is created
    - useIpAddress – specifies whether the channel should use IP addresses in URLs rather than hostname of the server
    - listen – setting for activation hooks into listener service
    - bindTo – used with computers that have multiple IP addresses
    - machineName – overrides useIpAddress
    - rejectRemoteRequests (tcp only) – sets local communication only

Configuration Files (cont)

- Providers
  - Sink and formatter providers allow the user to specify the manner in which messages are generated and captured by the framework for each channel.
  - Both the client and server may specify settings for
  - The tags <serverProviders></serverProviders> and <clientProviders></clientProviders> contain the individual settings for each provider or formatter that you wish to set.
  - You can specify one formatter and multiple provider settings.
  - You must place the settings in the order shown:

Configuration Files (cont)

- Example channel entry for a server:

```
<channels>
    <channel ref=“http” port=“1234”>
        <serverProviders>
            <formatter ref=“binary” />
            <provider type=“MySinks.Sample, Server” />
        </serverProviders>
    </channel>
</channels>
```

Configuration Files (cont)

- Providers (cont)
  - Available attributes for formatters and providers:
    - ref – “soap”, “binary”, or “wsdl”
    - type – if ref is not specified, contains namespace, classname, and assembly of the sink provider implementation.
    - includeVersions (formatter only) – specifies whether version information is included with object requests
    - strictBinding (formatter only) – specifies whether the server must use an exact type and version for object requests

Configuration Files (cont)

- Service
  - The <service> tag is used in the server's configuration file to specify the remote objects that will be hosted.
  - Contains <wellknown /> and <activated /> entries for server-activated objects (SAOs) and client-activated objects (CAOs), respectively.
  - Valid attributes for <wellknown />
    - type – Specifies the namespace, classname, and assemblyname of the remote object.
    - mode – Singleton or SingleCall
    - objectUri – Important for IIS hosting (URIs must end in .rem or .soap, as those extensions can be mapped into the IIS metabase.
    - displayName – Optional, used by .NET Framework configuration tool.
  - Valid attributes for <activated />
    - type – Specifies the namespace, classname, and assemblyname of the remote object.

Configuration Files (cont)

- Client
  - The <client> tag is used in the client's configuration file to specify the types of remote objects that it will use.
  - Contains attribute for the full URL to the server if using CAOs.
  - Contains <wellknown /> and <activated /> entries for server-activated objects (SAOs) and client-activated objects (CAOs), respectively.
  - Valid attributes for <wellknown />
    - url – The full URL to the server's registered object
    - type - Specifies the namespace, classname, and assemblyname of the remote object.
    - displayName – Optional, used by .NET Framework configuration tool
  - Valid attributes for <activated />
    - type – Specifies the namespace, classname, and assemblyname of the remote object.

Configuration Files (cont)

- Usage notes:
  - Errors in your configuration file cause the framework to instantiate a local copy of the remote object rather than a proxy when you call new on it. Check the IsTransparentProxy method to be sure you are using a remote object.
  - When you specify assembly names in your <wellknown /> and <activated />, don’t include the extension (.dll or .exe).
  - You only have to specify the features that you want/need in your configuration file.
  - You don’t have to use the <channel /> setting on the client if you use the default “http” or “tcp” channels on the server. You must specify a port on the server.

Soapsuds and Implementation Hiding

- The first thing that you may notice when using .NET remoting is that the remote object assemblies must be present on the client.
- We can get away with using interfaces to hide implementation if we stick with programmatic remoting configuration.
  - Create an assembly that contains interfaces which can be included on the client machine.
  - Create another assembly which contains the remote object implementations of the interfaces you specified earlier.
  - Call Activator.GetObject on the client when you want a class that implements the interface you specify.

Soapsuds and Implementation Hiding (cont)

- Example:

In the shared assembly:

```
public interface IExampleClass {…}
```

In the assembly on the server:

```
public class ExampleClass : MarshalByRefObject, IExampleClass {…}
```

On the client:

```
IExampleClass iec = (IExampleClass) Activator.GetObject(
                                    typeof(IExampleClass),
                                    “tcp://localhost:1234/ExampleClass”);
```

Soapsuds and Implementation Hiding (cont)

- Soapsuds is a Visual Studio tool that allows you to extract metadata from an assembly. The new assembly contains no implementation detail, just meta (type) information.
- If our application contains only remote objects and no customized [serializable] objects, then we can just run soapsuds on the assembly containing our remote objects and include the new assembly on the client.

```
soapsuds -ia:MyRemoteObjects -nowp -oa:MyRemoteMeta.dll
```

Soapsuds and Implementation Hiding (cont)

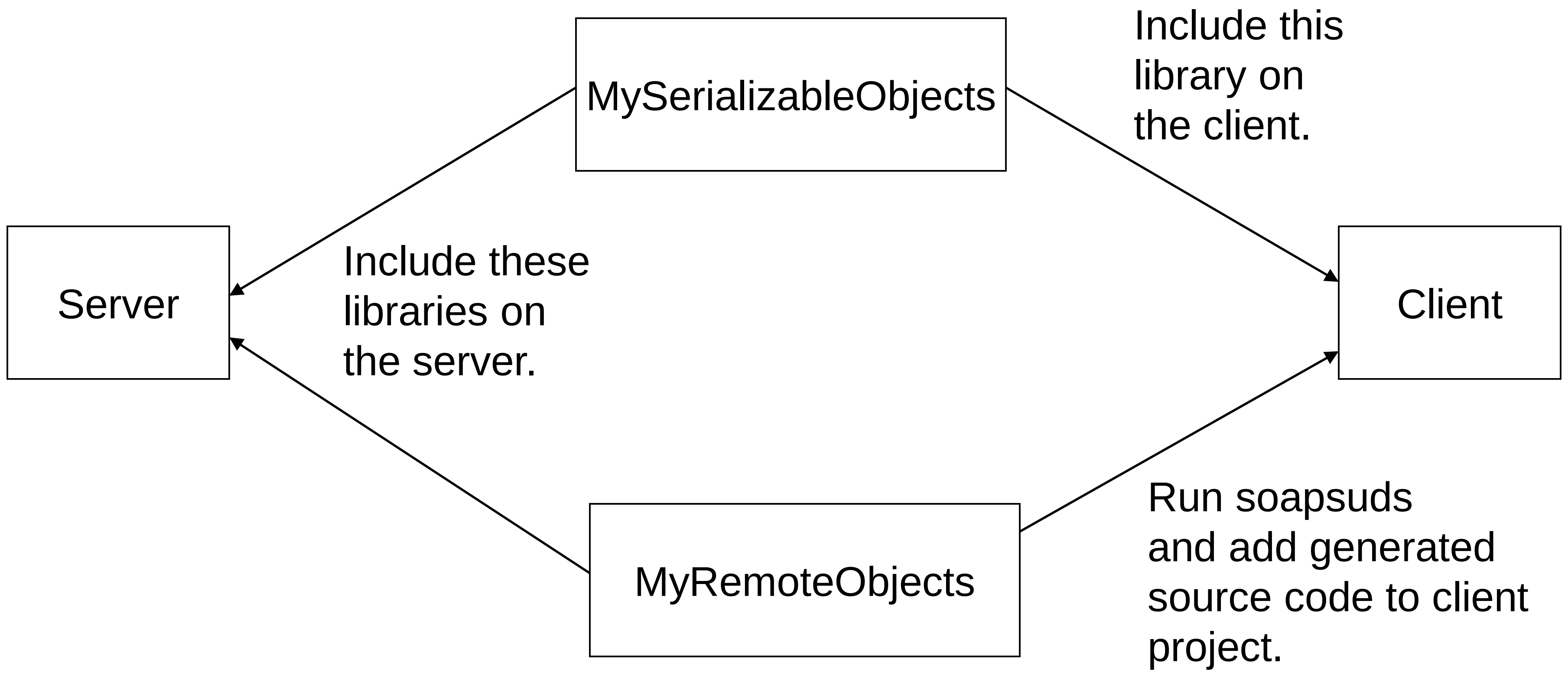
- If our application does include custom [serializable] objects that are passed between domains, then we can't just generate a new assembly. We have to use generated source code that describes the remote object metadata.

  soapsuds -ia:MyRemoteObjects –nowp –gc

- Note that this does not include objects that are native to the framework, i.e. strings, FileInfo, DirectoryInfo, etc. If our application only uses these types of [serializable] objects, then generating a "meta" assembly will work fine.

Soapsuds and Implementation Hiding (cont)

Project configuration:



MySerializableObjects
Include this library on the client.
Server
Include these libraries on the server.
Client
MyRemoteObjects
Run soapsuds and add generated source code to client project.

Server Deployment with Windows Services

- A .NET windows service inherits from System.ServiceProcess.ServiceBase
- Place your application specific code in the OnStart(..) method.
- You have to provide an installer class along with your windows service class.
- Using a windows service allows you to do event logging
- If your service does remoting, you have to place the configuration file in c:\WINNT\system32
- Install the service using *installutil YourServiceName.exe*
- After you’ve installed the service, you can start it using the Microsoft Management Console.

Server Deployment with IIS

- If you are concerned about security, then IIS hosting is the best way to go.
- Authentication and encryption features are available through IIS.
- Remote objects are now hosted in IIS; there is no Main() in the server.
- Updates to the server are easy: just copy over the remote object assembly and web.config file. IIS will automatically read the new data.

Server Deployment with IIS

- Procedure:
  - Create a class library for your remotable objects
  - Build the assembly for the class library
  - Create a web.config file for the server
  - Create a virtual directory on the host machine
  - Set the desired authentication methods for the directory
  - Place the web.config file in the virtual directory
  - Create a /bin directory in the virtual directory
  - Place the remotable object assembly in the virtual directory
  - Create a client and configuration file

Client Deployment with IIS

- By placing a WinForm application in a virtual directory, we can stream it to clients.
- When a URL is selected by a client machine, an HTTP request is sent to the server, which streams the application back to the client.
- The application is then stored in the browser cache and also the .NET download cache.
- The runtime opens the application automatically and also makes requests for additional assemblies and files as necessary.
- Be sure to put any remoting configuration files in the virtual directory with the client application.

End of Presentation