.NET Remoting

Prof Bill Buchanan
Http://www.dcs.napier.ac.uk/~bill/e_presentations/dotnet_remoting/dotnet_remoting.htm
In-the-cloud computing

Services:
- Identity service
- Email service
- Web service
- Disk storage service
- Application service
- Processing service

Users:
.NET Remoting v. Web Services

- Client
- Proxy
- .NET Remoting
- Listener
- Remote Objects (DLL)
- Remote Objects (DLL)
- Client
- Proxy
- Web Services
- Web services

Author: Bill Buchanan
.NET Remoting

Proxy (using SOAPSUDS or Add Reference)

Remote Objects (DLL)

Client communicates with the local proxy which hides the complexity of the connection

Web Services

Proxy (using wsdl or Add Reference)

Client

Listener

.NET Remoting v. Web Services
In general distributed systems are generally more scalable, more robust, and increase availability of services.

The most common distributed protocols are:

- **RPC (Remote Procedure Calls),**
- Microsoft Distributed Object Model (**DCOM**),
- Common Object Request Broker Architecture (**CORBA**) and
- Java Remote Invocation (**RMI**).
.NET Remoting allows access to objects placed around a network. It has:

- Communications channel.
- A formatter. This can be Binary encoded or XML encoded (SOAP).
Application Domain (Assembly)

Application

Local object

Communication channel

Formatter: Binary (fast, but not as compatible as XML-based)

Formatter: XML (highly compatible, but not as fast - SOAP)

Remote object

.NET Remoting

.NET Framework

Formatting and channel
Application Domain (Assembly)
Does the object have persistence?
public void myObjectProcessing (myObjectClass obj) {
}
Stateful firewall

.NET Remoting

Remote object

Application Domain (Assembly)

Object

Serialise

Encapsulate (SOAP)

Formatter (encoder)

Communicate

Serialise

Encapsulate (SOAP)

Communicate

Application
Client- and server-activated objects (CAO and SAO)
**Client-activated objects**
These have a finite lease time, and once their lease has expired they are deleted (using the garbage collector).

**Server-activated objects**
These can either be defined with a "single call" or with a "singleton".
- A **single call** accepts one request from a client, and then performs the action, and is then deleted (with the garbage collector). It is defined as stateless, as it does not hold onto parameters from previous calls.
- **Singletons** are stateful and can accept multiple calls, where they remember previous calls, and retain their information. They can thus communicate with multiple clients. Also the lifetime of singletons is controlled by lease-based lifetime.

The main namespaces used for remoting:
- **System.Net**. This includes classes relating to the networking elements of the distributed system.
- **System.Runtime.Remoting**. This includes classes for the remote aspects of the .NET framework, such as mechanisms for remote communication between objects.
- **System.Web.Services**. This includes protocols relating to Web services, such as those relating to HTTP and SOAP. This is defined as the ASP.NET Web services framework.
Objects passed by reference or value for local objects. For remote it is not possible to pass by reference – they must be **marshalled**.
public class ShowCapital : MarshalByRefObject
{
    public ShowCapital()
    {
    
    }
    public string show(string country)
    {
    
    }

TcpChannel channel = new TcpChannel(1234);
ChannelServices.RegisterChannel(channel);
Channels map onto TCP ports.
Standard ports:

21 – FTP.
23 – TELNET.
56 – DNS.
80 – WWW.
110 – POP-3

Over 1024 for development

ShowCapital()
Server activation is typically used when remote objects do not required to maintain their state between method calls ("single call"), or where there are multiple clients who call methods on the same object instance where the object maintains its state between function calls (singleton).

In a client-activated object, the client initiates the object and manages it for its lifetime.
Client application domain

Client object

Proxy

Remoting System

SAO – Single call

What’s the capital of Scotland?

Edinburgh

SAO - Singleton

Login name: Fred

OK

Send email To Bert

OK

CAO – Client manages lifetime

Get data

Get data

Get data

Delete
.NET Framework

Registering Remote Objects

Author: Prof Bill Buchanan
**Object mode.** This defines the server activation, such as SingleCall or Singleton.

**WellKnownObjectMode.SingleCall**

**Object URI.** This is the indicator that clients use to locate the object.

`newClass.ShowCapital`

**Assembly name.** This defines the assembly in which the class is contained in.

"ShowCapital1"

**Type name.** This defines the data type of the remote object.

`typeof(newclass.ShowCapital)`
Remote objects are registered using the RegisterWellKnownServiceType method by passing the required parameters into the method, such as:

```csharp
RemotingConfiguration.RegisterWellKnownServiceType (typeof(newClass.ShowCapital), "ShowCapital1", WellKnownObjectMode.SingleCall);
```

- **Assembly name.** This defines the assembly in which the class is contained in.
  "ShowCapital1"

- **Type name.** This defines the data type of the remote object.
  `typeof(newClass.ShowCapital)`

- **Object URI.** This is the indicator that clients use to locate the object.
  `newClass.ShowCapital`

- **Object mode.** This defines the server activation, such as SingleCall or Singleton.
  `WellKnownObjectMode.SingleCall`
RemotingConfiguration.Configure("myconfig.config");

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
          type="newclass.ShowCapital, newclass"
          objectUri="ShowCapital1" />
      </service>
      <channels>
        <channel ref="tcp server" port="1234" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

... otherwise we can use a config file
Once registered, the remote object will not instantiate itself, as this requires the client to initiate it, or call a method. This is achieved by a client which knows the URI of the remote object and by registering the channel it prefers using GetObject, such as:

```csharp
TcpClientChannel channel = new TcpClientChannel();
ChannelServices.RegisterChannel(channel);
ShowCapital sh = (ShowCapital)Activator.GetObject(
    typeof(newclass.ShowCapital),"tcp://localhost:1234/
    ShowCapital1");
```
Along with this, the compiler requires type information about the ShowCapital class when this client code is compiled. This can be defined with one of the following:

- With a reference to the assembly where the ShowCapital class is stored.
- Using **SOAPSUDS** tool to extract metadata directly from the endpoint. SOAPSUDS connects to the endpoint, and extracts the metadata, and generates an assembly or source code that is then used in the client compilation.
- By splitting the remote object into an implementation and interface class and then use the interface as a reference when compiling the client.
Creating a Remoteable Class
using System;
using System.Data;
	namespace newclass
{
    public class ShowCapital : MarshalByRefObject
    {
        public ShowCapital()
        {
        }
        public string show(string country)
        {
            if (country.ToLower()=="england")
                return("London");
            else if (country.ToLower()=="scotland")
                return("Edinburgh");
            else return("Not known");
        }
    }
}
Remoteable class
(newclass.dll)

ShowCapital()

System.Runtime.Remoting

Server application domain

Remoteable object
namespace newClass
{
    public class ShowCapital : MarshalByRefObject
    {
        public ShowCapital()
        {
            public string show(string country)
            {
                if (country.ToLower() == "england") return "London";
                else if (country.ToLower() == "scotland") return "Edinburgh"
                else return "Not known";
            }
        }
    }
}
Stateful firewall activation

ShowCapital()

Remoteable object (newclass.dll)

Remoting System
... otherwise we can use a config file
Configuration Files for Remoting
RemotingConfiguration.Configure( "myconfig.config");
RemotingConfiguration.Configure("app2.config");
sh = new ShowCapital();

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.runtime.remoting>
        <application>
            <client>
                <wellknown
                    type="newclass.ShowCapital, newclass"
                    url="tcp://localhost:1234/ShowCapital" />
            </client>
        </application>
    </system.runtime.remoting>
</configuration>
Stateful firewall

Marshalling

Client application domain

Server application domain

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton" type="newclass.ShowCapital, newclass"
          objectUri="ShowCapital1" />
      </service>
      <channels>
        <channel ref="tcp server" port="1234"/>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown type="newclass.ShowCapital, newclass"
          url="tcp://localhost:1234/ShowCapital" />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
using System;
using System.Data;
using IDCapital;

namespace newclass
{
    public class ShowCapital : MarshalByRefObject, IDCapital.IDCap
    {
        public string show(string country)
        {
            if (country == null)
                return "null";
            else if (country.Length <= 1)
                return country.ToUpper();
            else return country.Substring(0, 1).ToUpper() + country.Substring(1);
        }
    }
}

namespace newclass2
{
    class Class1
    {
        static void Main(string[] args)
        {
            TcpServerChannel channel = new TcpServerChannel(1234);
            ChannelServices.RegisterChannel(channel);
            RemotingConfiguration.RegisterWellKnownServiceType
                (typeof(newclass.ShowCapital), "ShowCapital",
                WellKnownObjectMode.SingleCall);
            Console.WriteLine("Starting...");    
            Console.ReadLine();
        }
    }
}

namespace IDCapital
{
    public interface IDCap
    {
        string show(string str);
    }
}
using System;
namespace IDCapital
{
    public interface IDCap
    {
        string show(string str);
    }
}

IDCap sh;

private void button1_Click(object sender, System.EventArgs e)
{
    string country, cap;
    country = textBox1.Text;
    cap = sh.show(country);
    textBox2.Text = cap;
}

private void Form1_Load(object sender, System.EventArgs e)
{
    TcpClientChannel channel = new TcpClientChannel();
    ChannelServices.RegisterChannel(channel);
    sh = (IDCapital.IDCap) Activator.GetObject(typeof(IDCap),
        "tcp://localhost:1234/ShowCapital");
}
soapsuds -url:http://localhost:1234/ShowCapital?wsdl
-oa:newclass.dll -nowp

Limitation:
Can only be used with HTTP and a Web Server
.NET Framework

Cram....

Author: Prof Bill Buchanan
- Client object requests an instance of the server object. The remoting system creates a proxy of the server object on the client side.
- The client calls a method on the server object, and the proxy sends the call through. The remoting system then sends a call to the server remoting system on the server.
- The remoting system on the server receives the call, and invokes the object on the server.
- The remote object on the server returns back the result to the remoting system on the server, which sends it to the remoting system on the client.
- The remoting system on the client sends result to the proxy which forwards it to the client object.
Object marshaling: defines how a remote object is exposed to client applications

Marshal-by-value (MBV) objects: These are when objects are copied and passed from the server application domain to the client application domain.

Which best defines a Marshal-by-reference (MBR) objects: These are when a proxy is used to access a server object, where the client keeps references to the objects.

Marshal-by-value objects: They reside on the server, and are serialized, and sent over the network to the client. The access is then done locally on the client.

Marshalk-by-value object:

```csharp
[Serializable()]
public class MyRemoteObject
{
    // ...
}
```

Marshalk-by-reference:

```csharp
public class MyRemoteObject: MarshalByRefObject
{
    // ...
}
```
Small objects, which type is likely to be the fastest: **Marshal-by-value**

Large objects, which type is likely to be the fastest: **Marshal-by-reference**

Objects which are only available on the server: **Marshal-by-reference**
Which is the most efficient methods of channels and formatting:
TCP channel, with a binary formatter

Which is the least efficient methods of channels and formatting:
HTTP channel, with a SOAP formatter

Which is the best methods of channels and formatting for interoperability:
HTTP channel, with a SOAP formatter

Which is the worst methods of channels and formatting for interoperability:
TCP channel, with a binary formatter

Default: TCP is BINARY
Default; HTTP is SOAP

The lifetime of the SAO is controlled by the server, while the lifetime of a CAO is controlled by the client
Conclusions
.NET Remoting:
- Highly extensible for all .NET components
- Interrupting SOAP messages

Web Services:
- Stateless

Data types:
- Any data type
- Defined by XSD, and thus limited

Ease-of-development:
- Fairly complex
- Easy to create

Interoperability:
- Only .NET Web Services
- Heterogeneous

Reliability:
- Needs to be built-in
- Built on IIS

State:
- Stateful/stateless

Pros and cons:
- .NET Remoting v. Web Services

Web Services:
- Built on IIS

Author: Bill Buchanan
Stateful firewall

.NET Remoting

Web Services

Data Storage

Business Logic

HTTP/SOAP

Binary(/HTTP)

ASP.NET Web Service Wrapper

Intranet

Internet

Pros and cons

.NET Framework
Windows Communications Foundation
Service-oriented Architecture (Services consumed by consumers)

What’s required?
A (Address)
B (Binding) – Protocol SOAP over HTTP, SOAP over TCP, SOAP over MQ
C (Contract) – Each service exposes a contract via an endpoint. URL defined for endpoint.
Service-oriented Applications (SoA)

Service-oriented Architecture

- Boundaries are explicit.
- Services are autonomous.
- Services share schema and contract, not class.
- Service compatibility is determined based on policy.

WCF

.NET Framework

Distributed Systems
Channels, Bindings, Contracts, Behaviours, Addresses and Endpoints.

A service becomes a client. Dealt with by WCF.
Applications are located at this layer.

In this layer services describe themselves to clients. It defines data and message contracts and bindings/policies.

This defines the behaviours within services, which are loaded in this layer.

 Supports difference channels and encoders, and thus communication for services.

This can be a range of hosts, such as for Internet Information Services (IIS) and starts and stops your services automatically. Another is executable files (.EXE) for the services are started and stopped manually within the code.
myCountryServer.IsBackground = true;
myCountryServer.Start();

ChannelFactory<Tut1.Contract.IService> scf;
scf = new ChannelFactory<Tut1.Contract.IService>(
    new NetTcpBinding(),
    netTcp://localhost:8080);

Tut1.Contract.IService s;
s = scf.CreateChannel();

ServiceHost svh = new ServiceHost(typeof(ServiceImplementation));
svh.AddServiceEndpoint(
    typeof(Tut1.Contract.IService),
    new NetTcpBinding(),
    nettcp://localhost:8080);
svh.Open();

[ServiceContract]
public interface IService
{
    [OperationContract]
    public string GetCapital(string Country);
}
using System.ServiceModel;
namespace Tut1.Contrct
{
    [ServiceContract]
    public interface IService
    {
        [OperationContract]
        string Contact(string message);
    }

    class ServiceImplementation : Tut1.Contract.IService
    {
        public string Contact(string Country)
        {
            Console.WriteLine("Contacting server...");
            if (Country == "scotland") return ("edinburgh");
            else if (Country == "england") return ("london");
            return "Not known";
        }
    }
}

public class Program
{
    private static System.Threading.AutoResetEvent endServer =
        new System.Threading.AutoResetEvent(false);

    public static void Main()
    {
        ServiceHost svh = new ServiceHost(typeof(ServiceImplementation));
        svh.AddServiceEndpoint(
            typeof(Tut1.Contract.IService),
            new NetTcpBinding(), "net.tcp://localhost:8080");
        svh.Start();
        Console.WriteLine("Server running");
        endServer.WaitOne();
        Console.WriteLine("Server shutting down");
        svh.Close();
        Console.WriteLine("Server stopped");
    }
}

[ServiceBehavior(InstanceContextMode =
    InstanceContextMode.PerCall)]
class ServiceImplementation : Tut1.Contract.IService
{
    public string Contact(string Country)
    {
        Console.WriteLine("Contacting server...");
        if (Country == "scotland") return ("edinburgh");
        else if (Country == "england") return ("london");
        return "Not known";
    }
}

public class Program
{
    private static System.Threading.AutoResetEvent endServer =
        new System.Threading.AutoResetEvent(false);

    public static void Main()
    {
        ServiceHost svh = new ServiceHost(typeof(ServiceImplementation));
        svh.AddServiceEndpoint(
            typeof(Tut1.Contract.IService),
            new NetTcpBinding(), "net.tcp://localhost:8080");
        svh.Start();
        Console.WriteLine("Server running");
        endServer.WaitOne();
        Console.WriteLine("Server shutting down");
        svh.Close();
        Console.WriteLine("Server stopped");
    }
}
DLL