Practical Lecture 4
Introducing .NET Remoting

1

Practical Session Structure

1. Introduction
2. Building a business component
3. Building an admin GUI
4. **Introducing .NET remoting**
5. Creating a web service and client website
6. Developing a Java client

2

2

Overview

- In order to start this session, you need to have completed all of the practical lecture 3
- In this lecture we will take the business component built in practical lecture 2 and the GUI implemented in practical lecture 3 and we will make them communicate using .NET remoting

3

3

Learning Objectives

- Understand the concepts involved in .NET remoting
- Write code that implements .NET remoting

4

4

Introduction

- In this practical session we will:
  - Implement a remote server application
  - Use the remote server application to distribute the admin tool and the PTSLibrary component

5

5

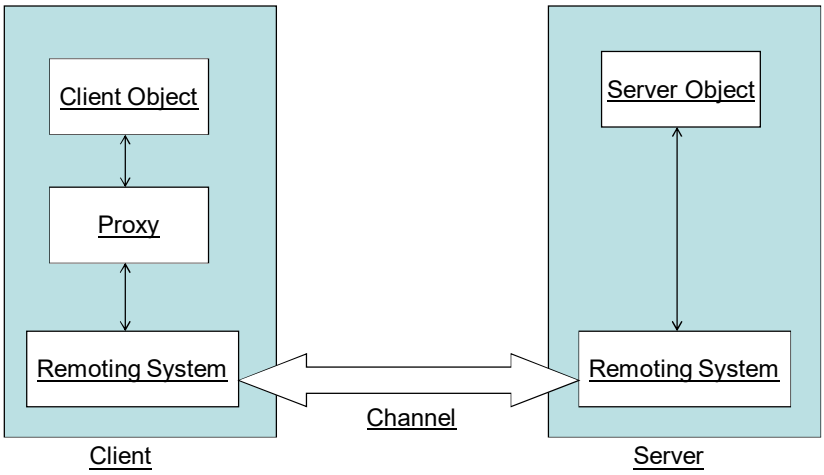
Distributed Systems in .NET

- There are two ways of building distributed systems in .NET, which provide means to invoke an object on another computer via a local proxy
  - Web Services: works across platforms, so can be used to provide services to clients that are not under your control and could be written in any language
  - .NET Remoting: works only when client and server are written in .NET. Can be used when both are under your the control

6

6

.NET Remoting Architecture



7

7

Marshalling

- Marshalling determines how an object is exposed to the client application
- Objects can be marshalled
  - By value: a copy of the server object is sent and kept in the client domain
  - By reference: the client only holds a reference to the object

8

8

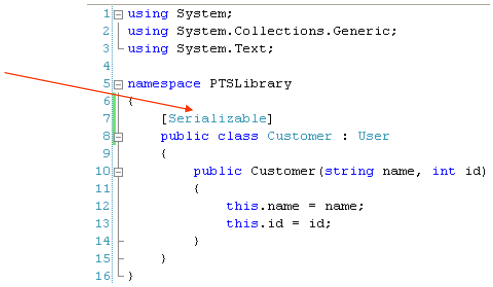
Marshal by Value

- In our application we will marshal the business objects by value
- The objects will then reside on the client and calls to them will be faster than marshalling by reference
- To do this, just add the [Serializable] attribute to the class that you want to marshal

9

Marshalling by Value /2

- Add [Serializable] to the following classes in the PTSLibrary project:
  - Customer
  - Project
  - Task
  - Team
  - TeamLeader
  - User

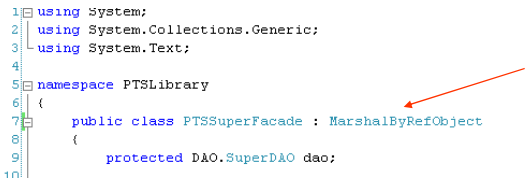


```
using System;
using System.Collections.Generic;
using System.Text;

namespace PTSLibrary
{
    [Serializable]
    public class Customer : User
    {
        public Customer(string name, int id)
        {
            this.name = name;
            this.id = id;
        }
    }
}
```

10

Marshalling by Reference

- We will marshal the *PTSAdminFacade* class by reference
- The object will reside in the server domain and environment and all calls are made via the proxy
- To do this, just make the class inheriting from *MarshalByRefObject* (as *PTSAdminFacade* already inherits from *PTSSuperFacade*, we can put the inheritance on the super class)



```
using System;
using System.Collections.Generic;
using System.Text;

namespace PTSLibrary
{
    public class PTSSuperFacade : MarshalByRefObject
    {
        protected DAO.SuperDAO dao;
```

11

11

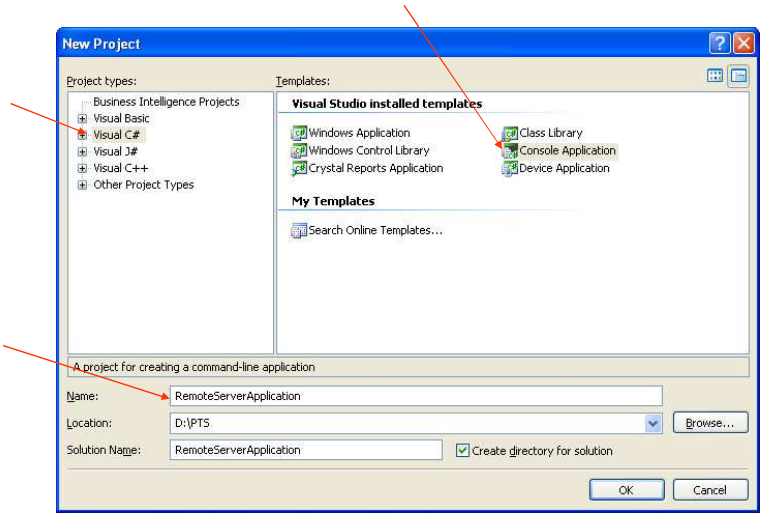
Creating the Project

- We will create the Admin GUI as a *Console Application* in a new solution
- Open Visual Studio 2005
- Go to *File -> New Project*
- Select *Visual C#* as the project type and then select *Console Application* as the template
- Name the project *RemoteServerApplication* and save it in a suitable location

12

12

Creating the Project /2



13

13

Remote Server Code

- By default, a *Program.cs* file is created
- Open this file and add the following code:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace RemoteServerApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            HttpChannel channel = new HttpChannel(50000);
            ChannelServices.RegisterChannel(channel, false);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(PTSLibrary.PTSAdminFacade),
                                                               "PTSAdminFacade", WellKnownObjectMode.Singleton);
            Console.WriteLine("Press the enter key to terminate server");
            Console.ReadLine();
        }
    }
}
```

14

14

Remote Server Code /2

- Things to note about the code:

The channel used is Http and port 50000.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace RemoteServerApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            HttpChannel channel = new HttpChannel(50000);
            ChannelServices.RegisterChannel(channel, false);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(PTSLibrary.PTSAdminFacade),
                                                               "PTSAdminFacade", WellKnownObjectMode.Singleton);
            Console.WriteLine("Press the enter key to terminate server");
            Console.ReadLine();
        }
    }
}
```

Register the channel with the Remoting framework

Ensures the console runs until the user presses enter

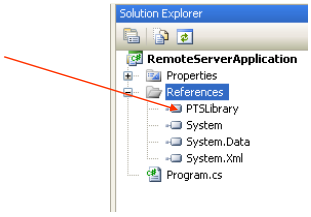
Register the remotable class (PTSAdminFacade) with the Remoting framework

15

15

Adding References

- As the remote server application provides access to classes from PTSLibrary, this project needs to be added as a reference



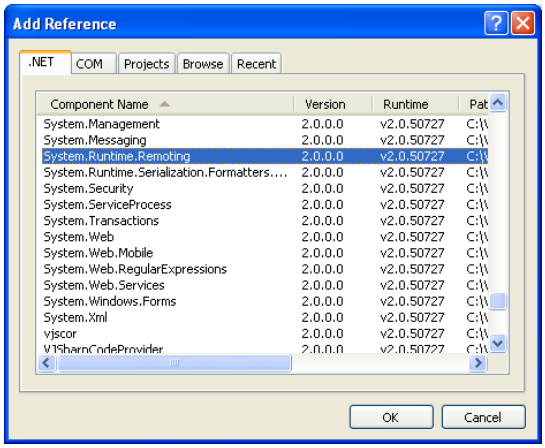
- Further, it is necessary to add a reference to the *System.Runtime.Remoting* namespace, as we need to access its classes for the .NET Remoting

16

16

Adding References /2

- Select *System.Runtime.Remoting* from the .NET tab



17

17

Adding the References /3

- Add the using directives to the class:

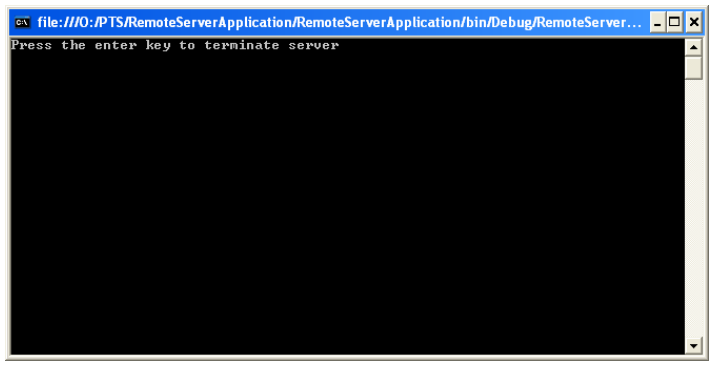
```
using System.Collections.Generic;
using System.Text;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using PTSLibrary;

namespace RemoteServerApplication
{
    class Program
    {
```

18

18

Run the Application

- Run the application and fix any problems there might be
- It should look somewhat like this:



19

19

Windows Firewall

- If Windows Firewall is running, it is possible that the server application is blocked when you try to run it
  - Make sure you unblock it

20

20

Changing the Admin Tool

- The Remote Server Application is ready
- We need to change the admin tool to work with .NET Remoting
- Open the *AdminApplication* project
- Start by adding the reference to *System.Runtime.Remoting* and add the using directives to the *frmAdmin* class
  - using System.Runtime.Remoting;
  - using System.Runtime.Remoting.Channels;
  - using System.Runtime.Remoting.Channels.Http;

21

21

Changing the Admin Tool /2

- The only other code that needs to change is when we instantiate *PTSAdminFacade* in the constructor of *frmAdmin*

```
public frmAdmin()
{
    InitializeComponent();
    HttpChannel channel = new HttpChannel();
    ChannelServices.RegisterChannel(channel, false);
    facade = (PTSAdminFacade)RemotingServices.Connect(typeof(PTSAdminFacade),
                                "http://localhost:50000/PTSAdminFacade");
    adminId = 0;
}
```

Similar as in the remote server application.

Creates a proxy

Denotes server

Denotes port

Denotes URI

22

22

Testing the .NET Remoting

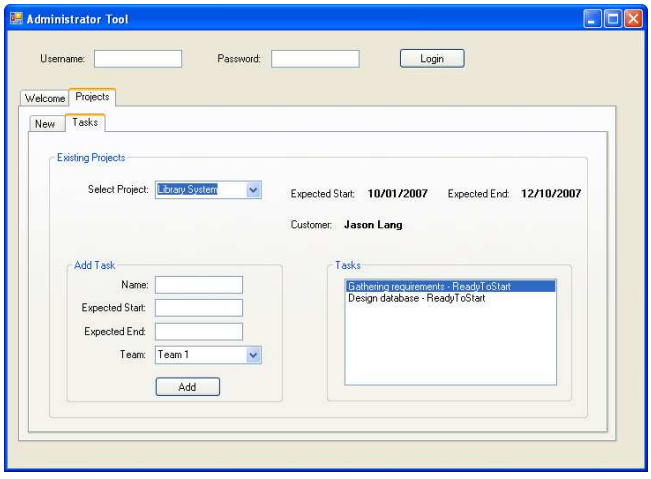
- Run the remote server application
- When the remote server application is running, start the admin tool
  - Check that everything still works
  - The fact that the admin tool is now communicating using .NET Remoting is completely transparent to the user, except that it might be a little slower

23

23

Testing the .NET Remoting /2

- When you run the admin tool now, it should behave just as before (maybe a little slower):



24

24

Summary

- In this session we have built a new project to act as a remote server application
- The business component was changed, so the required classes could be marshalled
- The existing admin tool was changed to work with .NET Remoting
- In the next session we will create a web service façade for the PTSLibrary and create the customer browser client that uses the web service

25