Software Architectures
Assignment 2
-
Calculator Web Service Architecture

David O Neill 0813001

November 15, 2011

Table Of Contents

1 Web Services

1.1 Support for the requirements

Service-oriented applications (SOA) that communicate across the web is a mechanism for supporting distributed computing with the intent of exposing services that typically cannot be exposed by conventional web implementations.

A conventional web implementation using the Hyper Text Transfer Protocol (http) is a mechanism to provide web content to web browsers. In the event of a developer wanting to communicate with a service using conventional 'application' means, such as method invocation, routines etc, an application server must be used. This application server handles the transfer of invocations and therein serialized information to the service or process which the application server exposes. This allows for almost transparent access to services resources by letting the application server do all the heavy lifting. Unlike in a typical situation of opening a socket and sender raw information down the socket.

For the assignment of a service computational calculator exposes an interface via a service definition contract or interface; defined in the Web Service Description Language (WSDL). This contract which clients subscribe provide the client with the means to interface and communicate correctly with the service.

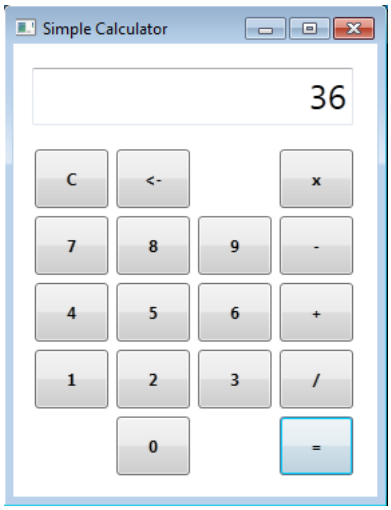

Fig. 1.1: Calculator

As the service implemented as web service, other clients can utilize the service as long as to conform to the specification of the contract. Orchestration can be catered for by design as the application and can be utilized as part of the web work flow.

2 Deployment Technologies

2.1 Visual Studio

Visual Studio an integrated development environment (IDE) is the axe of an serious wielding developer. This fully features integrated development environment (IDE) supports a multitude of different project types, catering for all possible scenarios for the current age. The editor itself supports all the features you would expect in an integrated development environment (IDE), such as code refactoring tools, autocompletion (intelliSense), class designers, code versioning and database schema designers, but takes it to a whole new level by the sheer attention to detail and implementation of these features make the environment a joy to work in and extremely productive.

2.2 C#

CSharp is a multi-paradigm language, to name a few imperative, declarative, functional and object oriented. The language developed by Microsoft is sworn by some to be the successor to Java. This strongly types language reintroduced many of the favored keywords such as virtual and such notable additions such as override which negate the need for interpretation and annotations seen in the java environment. C# itself is compiled into a command language runtime format like Java and as such can to be ran an any architecture that implements the .NET runtime. Such notable ports include Mono originally developed by Novel before going open source.

2.3 Windows Communication Foundation

2.3.1 Introduction

Windows Communication Foundation (WCF) is a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications (SOA) that communicate across the web. The implmentation of these applications is typically hosted on Internet Information Servers (IIS) which exposes the application to web requests. Windows Communication Foundation (WCF) realizes service oriented architectures principles to support clients in the distributed computing environment. The server implementation publishes it service for a Web Service Description Language(WSDL) typically in the Extendable Markup Language (XML) to which clients can subscribe to in order to consume or produce (send & receive) information to and from the service.

2.3.2 Service Contracts

In Windows Communication Foundation (WCF), service contracts are interfaces to which clients program to and service implementers realize. These interfaces are exactly the same as a normal interface in an object oriented language except special annotations [ServiceContract] & [OperationContract] are used to defined what will be exposed as part of the Web Service Description Language(WSDL) to which clients will subscribe to.

2.3.3 Data Contracts

In Windows Communication Foundation (WCF), data contracts are formal agreements between clients and the service regards data type, it's mechanism for defining abstract data types as the exchange information. A data contract is implemented as normal class to which special annotations are added, namely [DataContract] & [DataMember]. These annotations respectively define whether a class and its members should be exposed as part of the Web Service Description Language(WSDL) service contract.

2.4 Windows Presentation Foundation

2.4.1 Model View Controller

Windows Presentation Foundation Applications (WPF) is a model view controller User Interface Designer for rendering user interfaces. The separation of concerns primarily, business logic(controller) and the interface design(view). The controller can be written in any of the .Net family languages and the view is written in Xaml (XML). This allows clear separation of the interface from the business logic and allows different controllers written in different .NET languages to use the same interface consistently. Windows presentation foundation itself is a graphical subsystem built on top of DirectX allowing for greater efficiency over the old Graphical Device Interface (GDI) system.

2.4.2 View

The view in Windows Presentation Foundation Applications (WPF) is written in Extensible Application Markup Language (XAML) is a declarative language used in the specification of user interfaces comprised of controls such as buttons, text boxes, windows etc. The specification supports data binding, eventing and other features as a means to simplify the designing user interfaces. business logic

2.4.3 Controller

The controller in Windows Presentation Foundation Applications (WPF) can be written in any of the .NET family languages, its bound to the view at runtime and implements event handlers that accept the events that are generated for the view.

3 Development

3.1 Client

3.1.1 MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Client.ServiceReference1;
using System.Text.RegularExpressions;

namespace Client
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded( object sender , RoutedEventArgs e )
        {
            for( int i = 1 ; i < 21 ; i++ )
            {
                Object ctrl = this.FindName("button" + i);

                if( ctrl != null )
                {
                    Button b = ( Button ) ctrl;
                    b.Click += new RoutedEventHandler(b_Click);
                }
            }
        }


        private void b_Click( object sender , RoutedEventArgs e )
        {
            Button b = ( Button ) sender;
            string buttonAction = b.Content.ToString();

            switch( buttonAction )
            {
                case "0":
                case "1":
                case "2":
                case "3":
```

```
                case "4":
                case "5":
                case "6":
                case "7":
                case "8":
                case "9":
                case "+":
                case "-":
                case "x":
                case "/":
                    this.textBox1.Text += buttonAction;
                break;

                case "=":
                    this.compute();
                break;

                case "C":
                    this.textBox1.Text = "";
                break;

                case "<-":
                    this.undo();
                break;
            }

        }


        private void undo()
        {
            CalculatorClient client = new CalculatorClient();
            client.Open();
            EvaluatedExpression expr = client.undo();

            if( expr != null )
            {
                this.textBox1.Text = expr.Expression;
            }

            client.Close();
        }


        private void compute()
        {
            string expression = this.textBox1.Text;

            Regex pattern = new Regex( @"([\+-]?[0-9]+).*?(\+|-|x|/).*?([\+-]?[0-9]+)");

            MatchCollection mCollection = pattern.Matches(expression);

            if( mCollection.Count == 0 )
            {
                this.textBox1.Text = "Err";
                return;
            }

            CalculatorClient client = new CalculatorClient();
            client.Open();
```

```
115
116             foreach( Match m in mCollection )
117             {
118
119                 int lvalue = int.Parse( m.Groups[ 1 ].Value );
120                 string opt = m.Groups[ 2 ].Value;
121                 int rvalue = int.Parse( m.Groups[ 3 ].Value );
122
123                 EvaluatedExpression expr;
124
125                 switch( opt )
126                 {
127                     case "+":
128                         expr = client.opAddition( lvalue , rvalue );
129                     break;
130
131                     case "x":
132                         expr = client.opMultiplication(lvalue , rvalue);
133                     break;
134
135                     case "-":
136                         expr = client.opSubtraction(lvalue , rvalue);
137                     break;
138
139                     case "/":
140                         expr = client.opDivision(lvalue , rvalue);
141                     break;
142
143                     default:
144                         expr = null;
145                     break;
146                 }
147
148                 if( expr != null )
149                 {
150                     this.textBox1.Text = expr.Result.ToString();
151                 }
152             }
153
154             client.Close();
155         }
156     }
157 }
```

3.2 Server

3.2.1 ICalculator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace CalculatorService
{
    [ServiceContract]
    public interface ICalculator
    {
        [OperationContract]
        EvaluatedExpression opAddition( int lvalue , int rvalue );

        [OperationContract]
        EvaluatedExpression opSubtraction( int lvalue , int rvalue );

        [OperationContract]
        EvaluatedExpression opMultiplication( int lvalue , int rvalue );

        [OperationContract]
        EvaluatedExpression opDivision( int lvalue , int rvalue );

        [OperationContract]
        EvaluatedExpression undo();
    }

    [DataContract]
    public class EvaluatedExpression
    {
        string expression;
        int result = 0;

        [DataMember]
        public string Expression
        {
            get { return expression; }
            set { expression = value; }
        }

        [DataMember]
        public int Result
        {
            get { return result; }
            set { result = value; }
        }
    }
}
```

3.2.2 CalculatorService.svc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace CalculatorService
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
    public class CalculatorService : ICalculator
    {
        private static List<EvaluatedExpression> history;

        static CalculatorService()
        {
            CalculatorService.history = new List<EvaluatedExpression>();
        }

        public EvaluatedExpression opAddition( int lvalue , int rvalue )
        {
            EvaluatedExpression exp = new EvaluatedExpression();
            exp.Result = lvalue + rvalue;
            exp.Expression = lvalue + "+" + rvalue;
            CalculatorService.history.Add(exp);
            return exp;
        }

        public EvaluatedExpression opSubtraction( int lvalue , int rvalue )
        {
            EvaluatedExpression exp = new EvaluatedExpression();
            exp.Result = lvalue - rvalue;
            exp.Expression = lvalue + "-" + rvalue;
            CalculatorService.history.Add(exp);
            return exp;
        }

        public EvaluatedExpression opMultiplication( int lvalue , int rvalue )
        {
            EvaluatedExpression exp = new EvaluatedExpression();
            exp.Result = lvalue * rvalue;
            exp.Expression = lvalue + "x" + rvalue;
            CalculatorService.history.Add(exp);
            return exp;
        }

        public EvaluatedExpression opDivision( int lvalue , int rvalue )
        {
            EvaluatedExpression exp = new EvaluatedExpression();
            exp.Expression = lvalue + "/" + rvalue;

            if( lvalue == 0 || rvalue == 0 )
            {
                return exp;
            }

            exp.Result = lvalue / rvalue;
            CalculatorService.history.Add(exp);
            return exp;
        }

        public EvaluatedExpression undo()
        {
            if( history.Count == 0 )
            {
                return null;
            }

            EvaluatedExpression temp = history.ElementAt(history.Count - 1);
            history.RemoveAt(history.Count - 1);
            return temp;
        }
    }
}
```

3.3 Published Metadata

3.3.1 app.config (WSDL)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding
          name="BasicHttpBinding_ICalculator"
          closeTimeout="00:01:00"
          openTimeout="00:01:00"
          receiveTimeout="00:10:00"
          sendTimeout="00:01:00"
          allowCookies="false"
          bypassProxyOnLocal="false"
          hostNameComparisonMode="StrongWildcard"
          maxBufferSize="65536"
          maxBufferPoolSize="524288"
          maxReceivedMessageSize="65536"
          messageEncoding="Text"
          textEncoding="utf-8"
          transferMode="Buffered"
          useDefaultWebProxy="true">
          <readerQuotas
            maxDepth="32"
            maxStringContentLength="8192"
            maxArrayLength="16384"
            maxBytesPerRead="4096"
            maxNameTableCharCount="16384" />
          <security
            mode="None">
            <transport
               clientCredentialType="None"
               proxyCredentialType="None"
               realm="" />
            <message
              clientCredentialType="UserName"
              algorithmSuite="Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
    <endpoint
      address="http://localhost:1042/CalculatorService.svc"
      binding="basicHttpBinding"
      bindingConfiguration="BasicHttpBinding_ICalculator"
      contract="ServiceReference1.ICalculator"
      name="BasicHttpBinding_ICalculator" />
    </client>
  </system.serviceModel>
</configuration>
```

4 Evaluation

Fine application, works well and as intended, has state to support undo operations and no need for a database. Furthermore the server remembers its state event after a restart allowing for transactions to be completed successfully.

Unlike the java alternative the terminology used in the implementation technologies is clear and well understood. A developer with an understanding of web terminology, and object oriented terminology can surmise a great deal from the example.

The implementation characteristics of the Windows Communication Foundation application services and Web Service Description Language (WSDL) and are clear and concise allowing for the developer to focus on the objective and not on obscure non de-script naming conventions as seen in some of the alternative technologies.

The example application adequately identifies the key terminology present in application servers, contract, contract operations, data contracts and members and Web Service Description Language (WSDL).