Java Web Services and Integration with .NET

by
Venkat Subramaniam
venkats@durasoftcorp.com

May 2003

Presentation and examples can be downloaded from
http://www.agiledeveloper.com/download.aspx

Abstract

- Web Services are gaining a lot of popularity and have the potential to change the way we develop applications in the future. Web Services are based on standards and provide greater interoperability than the technologies of the past. Well known infrastructures include the ones from Sun and from Microsoft. These development kits improve the ease and speed of coding; however, in order to develop and especially debug a serious application, further understanding is essential. This talk will first focus on providing you a better understanding of the Java Web Services architecture. It will then present the classes involved, and details on what gets marshaled between the client and the server? It finally shows you how to integrate with the .NET Web Services in terms of communication and exception handling.

- Dr. Venkat Subramaniam is an agile developer who teaches and mentors software developers. He has significant experience in architecture, design and development of distributed object systems. Venkat has trained more than 2500 software professionals around the world. He is also an adjunct professor at University of Houston and teaches the Professional Software Developer Series at Rice University's Technology Education Center.

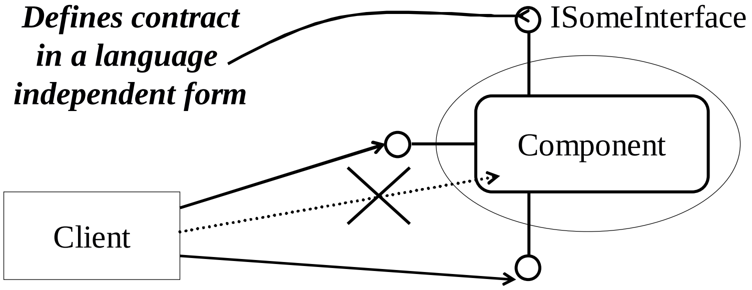
- Each page with a ★ has an attached example

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

Distributed Computing With Interfaces

- DCOM, CORBA, RMI architectures



Defines contract in a language independent form

Encapsulation
Flexibility to change implementation at will
Polymorphism and substitutability
Language interoperability
Platform interoperability(?)
Process/processor independence

Distributed Computing & XML

- Problems with DCOM & CORBA
  - – Interfaces based clients bind to the interface IDL
  - – DCOM works only among select platforms
    - Predominantly on Windows NT, 2000, etc.
  - – Client side ORB is required in case of CORBA
  - – **These technologies are for intranet and not internet**
    - firewall will block the requests from a truly remote client!
- XML
  - – provides cross platform transfer mechanism
  - – has multi-language / multi-vendor support
  - – So why not use XML to transport method invocation rather than simply data?

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

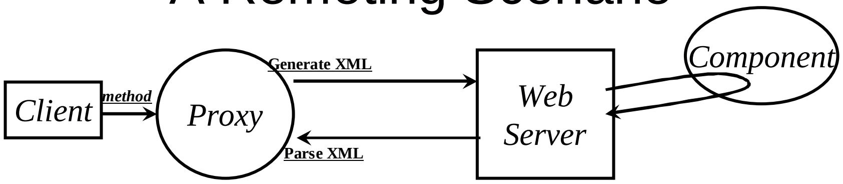
Why XML?

- XML is about extensibility and flexibility
- tags describe and surround the data
- Example:

```
<?xml version = "1.0" ?>
<equipment>
   <pump>
         <name> p01 </name>
         <pressure units="psi"> 32.23 </pressure>
   </pump>
   <pump>
         <name> p02 </name>
         <pressure units="psi"> 22.887 </pressure>
   </pump>
</equipment>
```

- Open, extensible
- Platform independent
- Self describing data
    - Data Exchange
- Supports query and discovery of data
- ***Dynamic Data Exchange***

A Remoting Scenario



- The Payload
    - XML document could simply present method names
        - as elements
    - data may be carried by sub-elements
- The receiving system could
    - parse the XML document
    - Generate an XML response and send it back to the client
- The client waits for response, receives & parses it

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

Sounds good, how do we do this?

- Each one inventing there own way to do this
  - Leads to duplication of effort
  - Re-inventing the wheel results in higher cost
  - What about efficiency?
- Standardization is essential

SOAP

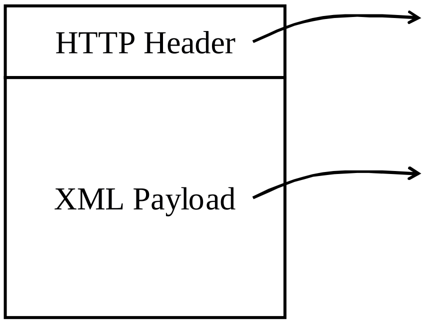
- A well-formed XML may be used as a wire protocol
- XML documents can be easily transported to remote systems
  - Most convenient to use HTTP (walks through firewall)
  - Other mechanisms like SMTP
- Mechanism is addressed by SOAP
  - **Simple Object Access Protocol** (SOAP)

- SOAP relies on the XML schema definition and elements
  - It uses the schema definition
  - The elements are used to communicate information
  - attributes are used to communicate SOAP specific meta-information

---

SOAP’s approach

- To use HTTP (or other mechanisms) for transport
- Use XML for data encoding/information exchange
- Keep it simple and extensible



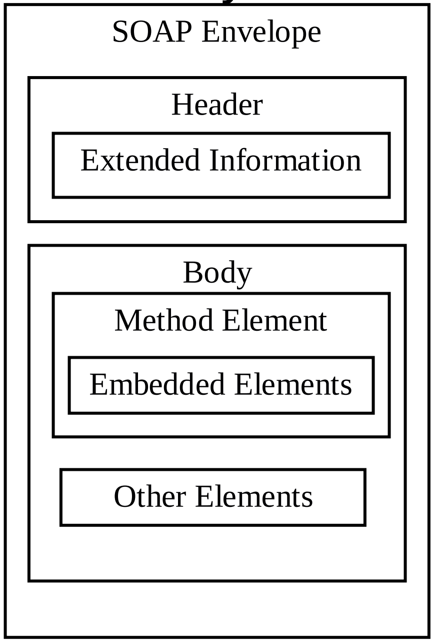
HTTP Method information

- Well-formed XML
- Strong data types enforced with Schema
- Details on method invocation

Why SOAP?

- Why not use existing transport mechanisms to achieve greater distribution across platforms?
- SOAP simply uses two existing and very popular standards
  - HTTP for transport protocol
  - XML for data interchange
- SOAP simply provides a interoperable wire protocol
- All you need is a HTTP server and an XML parse
  - Almost

Basic SOAP Payload Structure



SOAP Fault Response

- SOAP fault response provides following elements
  - faultcode
  - faultstring
  - runcode
  - details element (optional)

- These are part of *SOAP-ENV:Fault* element within the *SOAP-ENV:Envelope* return payload

---

SOAP Serialization

- Method call parameters and response are serialized
- SOAP specifies how each data type is serialized
  - It is standardized

- The rules predominantly come from XML Schema

- XML Schema provides structure for most standard data types:
  - string, boolean, float, int, dateTime, binary, etc.

- Objects are serialized as well as an XML element with child elements for the fields of the object

SOAP Serialization of Primitives

- Assume a method double getTemperature(String city)

method call structure

```
<soap:Envelope
        xmlns:xsi=
                "http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema“
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
                <soap:Body>
                        <getTemperature
                                xmlns="http://www.durasoftcorp.com/">
                                <city>Houston</city>
                        </getTemperature>
                </soap:Body>
</soap:Envelope>
```

SOAP Serialization of Primitives…

response structure

```
<soap:Envelope
        xmlns:xsi=
                "http://www.w3.org/2001/XMLSchema-instance“
        xmlns:xsd="http://www.w3.org/2001/XMLSchema“
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
                <soap:Body>
                        <getTemperatureResponse
                                xmlns="http://www.durasoftcorp.com/">
                                <getTemperatureResult>98.92
                                </getTemperatureResult>
                        </getTemperatureResponse>
                </soap:Body>
</soap:Envelope>
```

SOAP Serialization of Arrays

- Assume a method `int sum(int[] values)`

method call structure

```
<soap:Envelope …
            <soap:Body>
                  <sum …>
                        <values>
                              <int>2</int>
                              <int>5</int>
                        </values>
                  </sum>
            </soap:Body>
</soap:Envelope>
```

SOAP Serialization of Objects

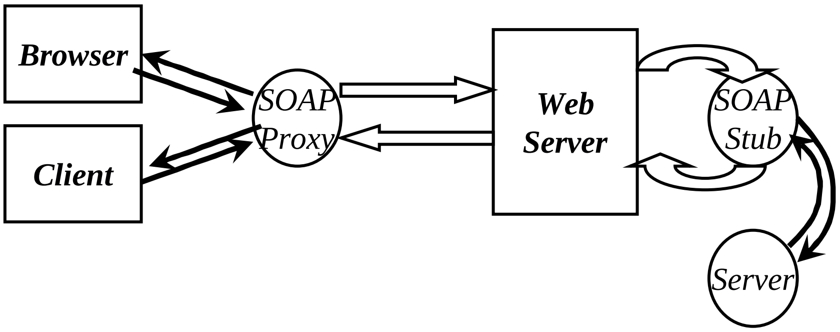
- Assume a method `void shipTo(Address addr)`

method call structure

```
<soap:Envelope …
            <soap:Body>
                  <shipTo …>
                        <address>
                              <street>101 Main St</street>
                              <city>Houston</city>
                              <state>TX</state>
                              <zip>77478</zip>
                        </address>
                  </shipTo>
            </soap:Body>
</soap:Envelope>
```

SOAP Applied

- Web Services is simply an application of SOAP



Web Service Infrastructure

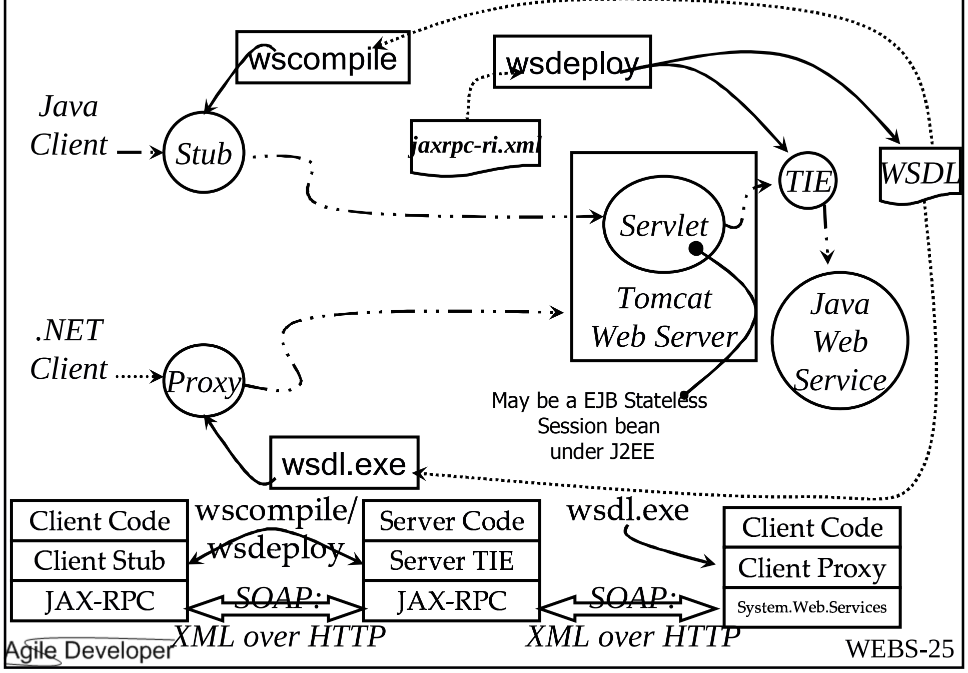
- Web Service Wire Format
  - Protocol to allow open communication between various entities independent of platform
  - SOAP
- Web Service Description
  - Allows for finding the details of web services, its methods, arguments, types, etc
  - Web Service Description Language (WSDL)
- Web Service Discovery
  - Allows for a web service to make its presence known and for a client to find it
  - SOAP Discovery (DISCO)
  - Universal Description, Discovery and Integration (UDDI)

Quiz Time!

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

Java Web Service and Clients



wscompile

wsdeploy

Java Client

Stub

jaxrpc-ri.xml

Servlet

Tomcat Web Server

TIE

WSDL

Java Web Service

.NET Client

Proxy

May be a EJB Stateless Session bean under J2EE

wsdl.exe

Client Code

Client Stub

JAX-RPC

wscompile/ wsdeploy

SOAP:

XML over HTTP

Server Code

Server TIE

JAX-RPC

wsdl.exe

SOAP:

XML over HTTP

Client Code

Client Proxy

System.Web.Services

Steps in writing a Java Web Service

- Methods of a Web Service have to be part of a remote interface
- Write a class to implement that interface
- Compile the Service Interface and Class
- Write a configuration file (jaxrpc-ri.xml) that indicates
  - the interface name
  - the class name
  - the service name
  - the target namespace
- Write a web.xml that ties the servlet and the web service
- Run mapping tool wsdeploy to generate server side *tie* class
- Deploy the web service

Writing a Java Web Service

```
package com.durasoftcorp.ws;
public interface TemperatureProviderIF
      extends java.rmi.Remote {
   public double getTemperature(String city)
      throws java.rmi.RemoteException;
}
```

```
package com.durasoftcorp.ws;
public class TemperatureProviderImpl
    implements TemperatureProviderIF,
    javax.xml.rpc.server.ServiceLifecycle {
      public void init(Object context){}
      public void destroy(){}
      public double getTemperature(String city)
      {
            // Ideally we would interface with a data
            // source that provides this information
            return Math.random() * 100;
      }
}
```

Preparing the Java Web Service

jaxrpc-ri.xml

```
<webServices …
  targetNamespaceBase="http://www.durasoftcorp.com/wsdl"
  typeNamespaceBase="http://www.durasoftcorp.com/types“>

  <endpoint
    name="TemperatureProvider" …
    interface="com.durasoftcorp.ws.TemperatureProviderIF"
    implementation=
      "com.durasoftcorp.ws.TemperatureProviderImpl"/>
  <endpointMapping endpointName="TemperatureProvider"
       urlPattern="/TemperatureProvider"/>
</webServices>
```

```
javac -d TemperatureProvider\WEB-INF\classes *.java
copy jaxrpc-ri.xml TemperatureProvider\WEB-INF
copy web.xml TemperatureProvider\WEB-INF
jar –cf WeatherInfo_predeploy.war WEB-INF (from TemperatureProvider)
wsdeploy –keep … WeatherInfo.war WeatherInfo_predeploy.war
```

Deploying the Java Web Service

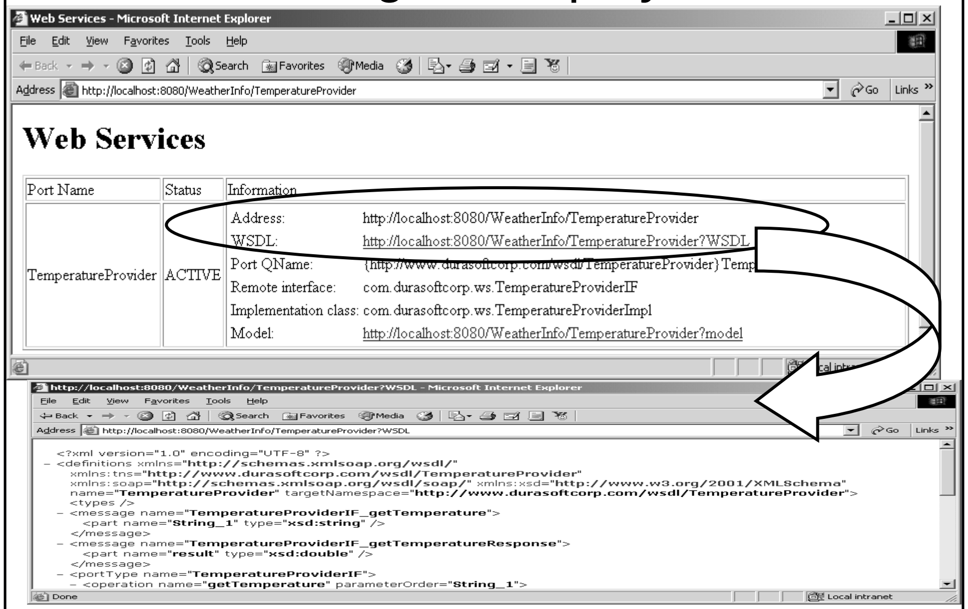
TemperatureProvider/WEB-INF/web.xml

```
…
<web-app>
  <display-name>
      Temperature Provider Application
  </display-name>
  <description>
      … return temperature for a city
  </description>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

wsdeploy fills in the rest of the details.
View the generated web.xml in the WeatherInfo.war

Drop the WeatherInfo.war into Tomcat's jwsdp-1.1/WebApps

Checking the deployment


Web Services - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/WeatherInfo/TemperatureProvider
Go Links

Web Services

Port Name	Status	Information	
TemperatureProvider	ACTIVE	Address:	http://localhost:8080/WeatherInfo/TemperatureProvider
		WSDL:	http://localhost:8080/WeatherInfo/TemperatureProvider?WSDL
		Port QName:	{http://www.durasoftcorp.com/wsdl/TemperatureProvider} Temp
		Remote interface:	com.durasoftcorp.ws.TemperatureProviderIF
		Implementation class:	com.durasoftcorp.ws.TemperatureProviderImpl
		Model:	http://localhost:8080/WeatherInfo/TemperatureProvider?model

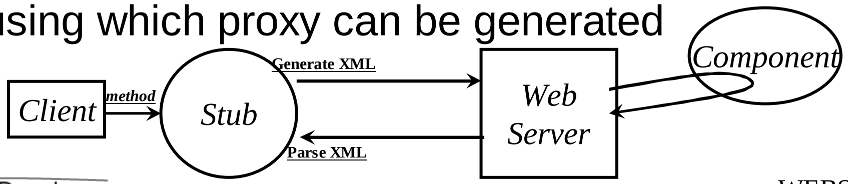
http://localhost:8080/WeatherInfo/TemperatureProvider?WSDL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/WeatherInfo/TemperatureProvider?WSDL
Go Links
Done
Local intranet

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.durasoftcorp.com/wsdl/TemperatureProvider"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="TemperatureProvider" targetNamespace="http://www.durasoftcorp.com/wsdl/TemperatureProvider">
    <types />
  - <message name="TemperatureProviderIF_getTemperature">
      <part name="String_1" type="xsd:string" />
    </message>
  - <message name="TemperatureProviderIF_getTemperatureResponse">
      <part name="result" type="xsd:double" />
    </message>
  - <portType name="TemperatureProviderIF">
    - <operation name="getTemperature" parameterOrder="String_1">
```

http://localhost:8080/WeatherInfo/TemperatureProvider

What about traditional clients?

- For non-web based client, you can create proxy
- The web client proxy receives a request, generates XML request & sends it using SOAP
- Receives response, parses it and hands it over to client!
- You can generate a web service proxy using the WSDL compiler.
- You provide a reference to service description using which proxy can be generated



Traditional Java client

Client.java main

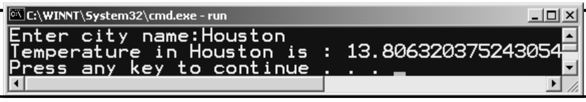
```
TemperatureProviderIF service =
     new TemperatureProvider_Impl()
          .getTemperatureProviderIFPort();

double temperature =
          service.getTemperature(city);
```

config.xml

```
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
  location=
     "http://localhost:8080/WeatherInfo/TemperatureProvider?WSDL"
  packageName="com.durasoftcorp.ws" />
</configuration>
```

```
wscompile –gen:client -keep –s keepDirName -d .  config.xml
javac Client.java
```

Generates stub from the WSDL



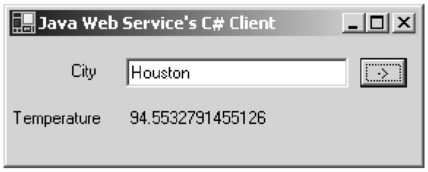
```
Enter city name:Houston
Temperature in Houston is : 13.806320375243054
Press any key to continue . . .
```

C# Client and C# Windows Client

Client.cs Main

```
WeatherInfo.TemperatureProvider service =
      new WeatherInfo.TemperatureProvider();

service.Url =
   "http://localhost:8080/WeatherInfo/TemperatureProvider";
double temperature = service.getTemperature(city);
```

```
wsdl /n:WeatherInfo http://localhost:8080/WeatherInfo/TemperatureProvider?WSDL

csc /out:Client.exe Client.cs TemperatureProvider.cs
```



City: Houston

Temperature: 94.5532791455126

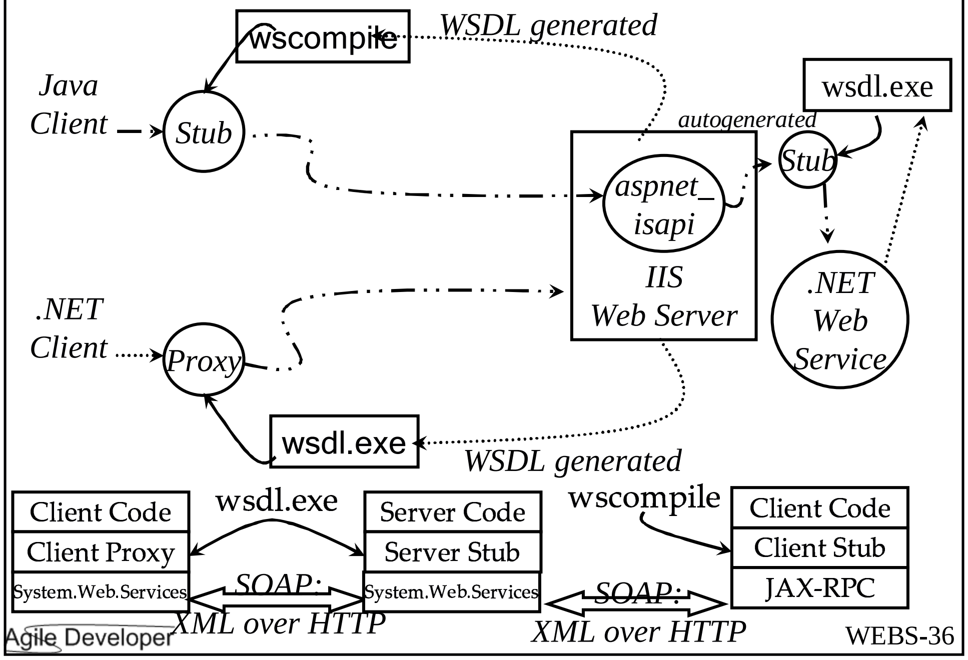
Generally no need to set
But (bug!) messes up port number
if port is not the default 80

Quiz Time!

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- ⇨ An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

.NET Web Service and Clients



wscompile

WSDL generated

Java Client

Stub

wsdl.exe

autogenerated

Stub

aspnet_ isapi

IIS Web Server

.NET Web Service

.NET Client

Proxy

wsdl.exe

WSDL generated

Client Code
Client Proxy
System.Web.Services

wsdl.exe

Server Code
Server Stub
System.Web.Services

SOAP:
XML over HTTP

wscompile

Client Code
Client Stub
JAX-RPC

SOAP:
XML over HTTP

Steps in writing a .NET Web Service

- Methods of a Web Service a marked using an attribute [WebMethod] – no separate interface
- Write a class to implement your Web Service methods with an extension asmx
  - No configuration files needed
  - No need to use any server side tools – done automatically
  - No deployment files needed
- Deploy the web service by copying to IIS directory

Writing a .NET Web Service

```
<%@ WebService Language="C#"
      Class= "TemperatureProvider" %>
using System;
using System.Web.Services;
[WebService(Namespace="http://www.durasoftcorp.com/ws")]
public class TemperatureProvider
{
    [WebMethod]
    public double getTemperature(string city)
    {
            // Ideally we would fetch this information
            // from our resources.
            return new Random().NextDouble() * 100;
    }
}
```

Preparing the .NET Web Service

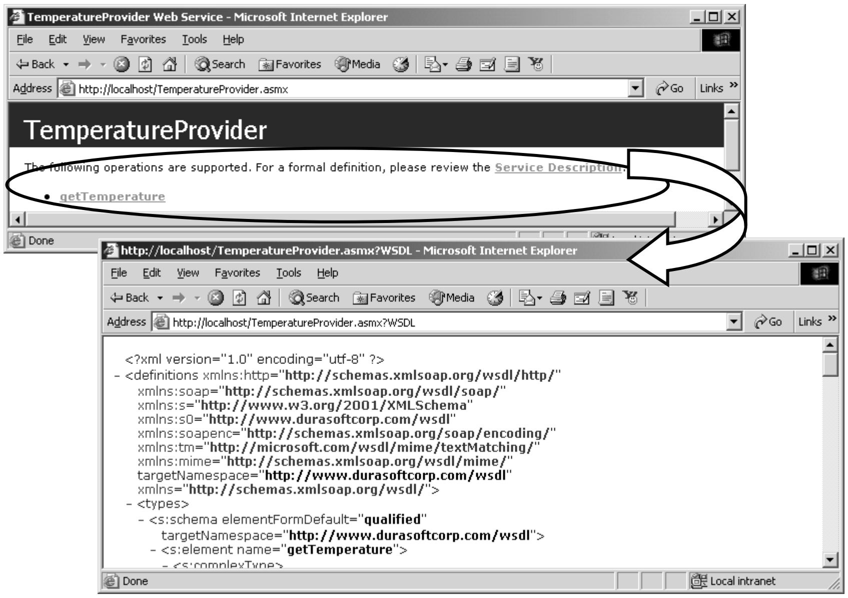
- Nothing to do in this step, the server side stubs are generated automatically
- Optionally, you could use the mapping tool to generate the server side stubs – no need

---

Deploying the .NET Web Service

Drop the TemperatureProvider.asmx file into inetpub\wwwroot

Checking the deployment


TemperatureProvider Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/TemperatureProvider.asmx
Go Links
Done
http://localhost/TemperatureProvider.asmx?WSDL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/TemperatureProvider.asmx?WSDL
Go Links
Done
Local intranet

TemperatureProvider

The following operations are supported. For a formal definition, please review the Service Description.

- getTemperature

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://www.durasoftcorp.com/wsdl"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://www.durasoftcorp.com/wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="http://www.durasoftcorp.com/wsdl">
      - <s:element name="getTemperature">
        - <s:complexType>
```

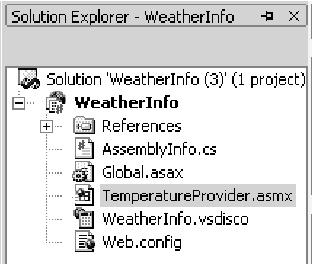
http://localhost/TemperatureProvider.asmx

Compiling the Web Service

- Packaging into DLL Module: useful to call from ASP.NET or rich clients
- Visual Studio .NET makes it easier to build a web service.
- Click on File, New, Project
- Select Visual C# Projects, and ASP.NET Web Service under Templates:.
- Modify Location to http://localhost/WeatherInfo
- Delete the created service1.asmx and create a new web service by right clicking WeatherInfo in solutions explorer, clicking on Add, Add Web Service.
- Give the name as TemperatureProvider.asmx and modify the code to add a method named temperature as shown in previous page.
- Right click on TemperatureProvider.asmx in solutions explorer and select “Set as start page.”
- Press Ctrl+F5 to test it!

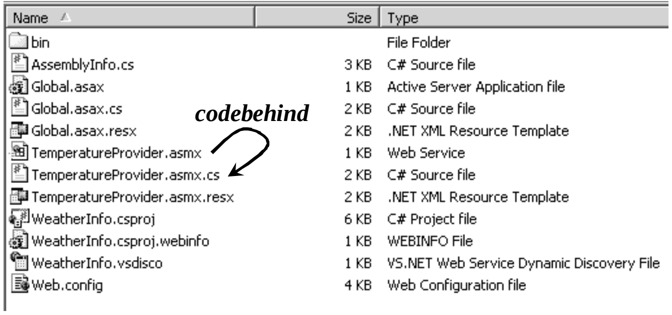
Code hidden by VS.NET!

- What you see is not what you have
- Visual Studio shows:



```
Solution 'WeatherInfo (3)' (1 project)
⊟ WeatherInfo
   ⊞ References
     AssemblyInfo.cs
     Global.asax
     TemperatureProvider.asmx
     WeatherInfo.vsdisco
     Web.config
```

- In reality you have:

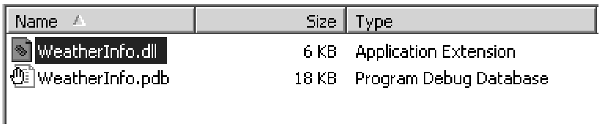


Name	Size	Type
bin		File Folder
AssemblyInfo.cs	3 KB	C# Source file
Global.asax	1 KB	Active Server Application file
Global.asax.cs	2 KB	C# Source file
Global.asax.resx	2 KB	.NET XML Resource Template
TemperatureProvider.asmx	1 KB	Web Service
TemperatureProvider.asmx.cs	2 KB	C# Source file
TemperatureProvider.asmx.resx	2 KB	.NET XML Resource Template
WeatherInfo.csproj	6 KB	C# Project file
WeatherInfo.csproj.webinfo	1 KB	WEBINFO File
WeatherInfo.vsdisco	1 KB	VS.NET Web Service Dynamic Discovery File
Web.config	4 KB	Web Configuration file

codebehind

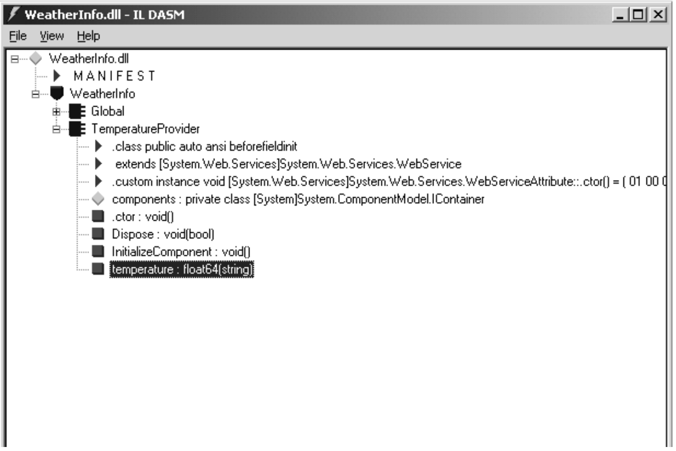
- Open these files in notepad and study!

Generated Assembly in bin



Name	Size	Type
WeatherInfo.dll	6 KB	Application Extension
WeatherInfo.pdb	18 KB	Program Debug Database

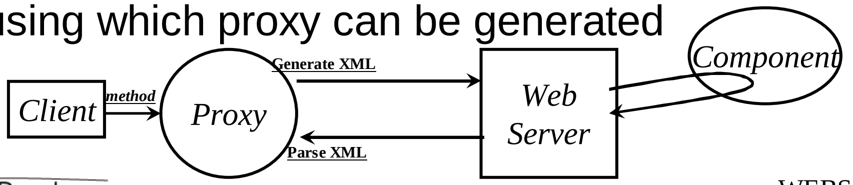
- We can open the assembly in ildasm



```
⊟ WeatherInfo.dll
   ▶ MANIFEST
 ⊟ WeatherInfo
   ⊞ Global
   ⊟ TemperatureProvider
       ▶ .class public auto ansi beforefieldinit
       ▶  extends [System.Web.Services]System.Web.Services.WebService
       ▶ .custom instance void [System.Web.Services]System.Web.Services.WebServiceAttribute::.ctor() = ( 01 00 0
       ◇ components : private class [System]System.ComponentModel.IContainer
       ■ .ctor : void()
       ■ Dispose : void(bool)
       ■ InitializeComponent : void()
       ■ temperature : float64(string)
```

What about traditional clients?

- For non-web based client, you can create proxy
- The web client proxy receives a request, generates XML request & sends it using SOAP
- Receives response, parses it and hands it over to client!
- You can generate a web service proxy using the WSDL compiler.
- You provide a reference to service description using which proxy can be generated



Traditional Java client

Client.java main

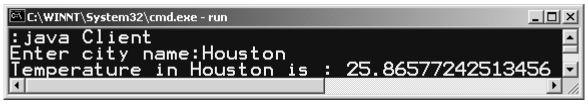
```
TemperatureProviderSoap service =
      new TemperatureProvider_Impl().
                  getTemperatureProviderIFSoap();

double temperature = service.getTemperature(city);
```

config.xml

```
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl
  location=
      "http://localhost/WeatherInfo/TemperatureProvider.asmx?WSDL"
  packageName="com.durasoftcorp.ws" />
</configuration>
```

```
wscompile –gen:client …  config.xml
javac Client.java
```

Generates stub from the WSDL



```
:java Client
Enter city name:Houston
Temperature in Houston is : 25.86577242513456
```

C# Client and C# Windows Client

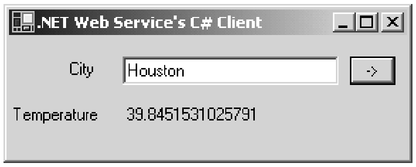
Client.cs Main

```
WeatherInfo.TemperatureProvider service =
      new WeatherInfo.TemperatureProvider();
```

~~`service.Url = "http://localhost/WeatherInfo/TemperatureProvider";`~~

```
double temperature = service.getTemperature(city);
```

```
wsdl /n:WeatherInfo   http://localhost/WeatherInfo/TemperatureProvider?WSDL

csc /out:Client.exe Client.cs TemperatureProvider.cs
```



City: Houston

Temperature: 39.8451531025791

Generally no need to set

Quiz Time!

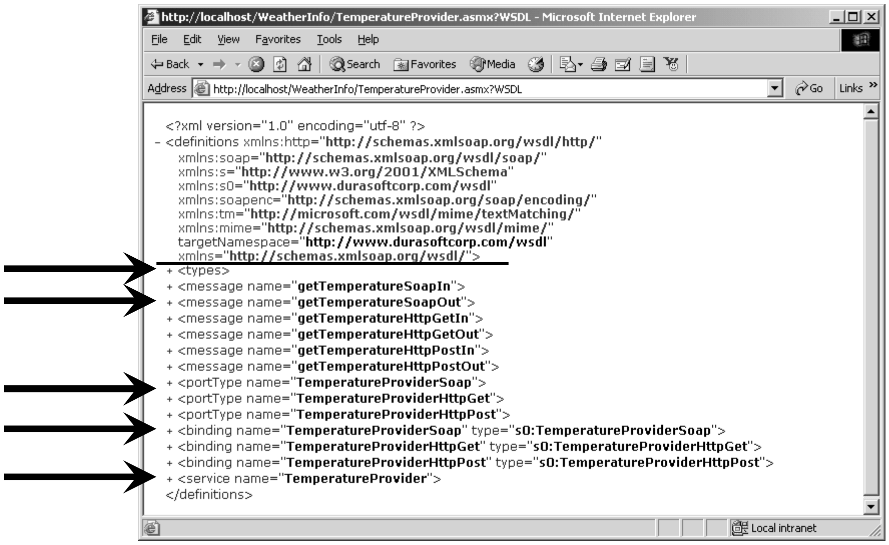
Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

WSDL

- Web Service Description Language
- Describes
  - the web service
  - the methods published by the web service
  - the arguments and results of methods
- Used to create the client proxy
- Description in the form of XML Schema
- Automatically generated using reflected metadata
  - kept in synch with the code changes
- You may view it by visiting your web service with a ?WSDL request
  - **http://localhost/WeatherInfo/TemperatureProvider.asmx?WSDL**

The WSDL contents

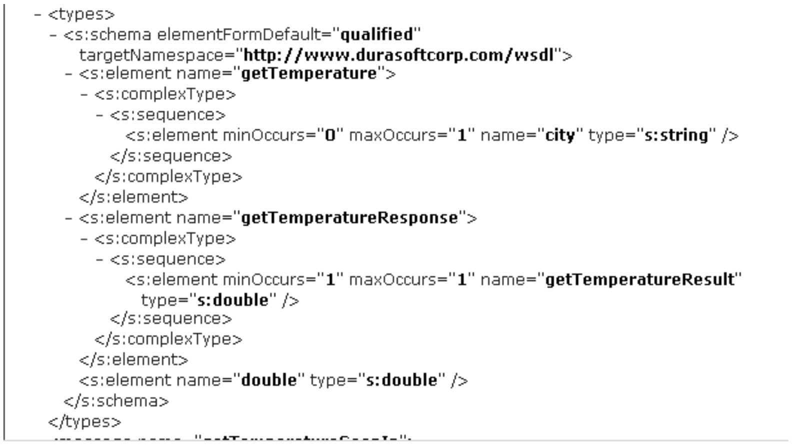


http://localhost/WeatherInfo/TemperatureProvider.asmx?WSDL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/WeatherInfo/TemperatureProvider.asmx?WSDL
Go Links
Local intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:s0="http://www.durasoftcorp.com/wsdl"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://www.durasoftcorp.com/wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  + <types>
  + <message name="getTemperatureSoapIn">
  + <message name="getTemperatureSoapOut">
  + <message name="getTemperatureHttpGetIn">
  + <message name="getTemperatureHttpGetOut">
  + <message name="getTemperatureHttpPostIn">
  + <message name="getTemperatureHttpPostOut">
  + <portType name="TemperatureProviderSoap">
  + <portType name="TemperatureProviderHttpGet">
  + <portType name="TemperatureProviderHttpPost">
  + <binding name="TemperatureProviderSoap" type="s0:TemperatureProviderSoap">
  + <binding name="TemperatureProviderHttpGet" type="s0:TemperatureProviderHttpGet">
  + <binding name="TemperatureProviderHttpPost" type="s0:TemperatureProviderHttpPost">
  + <service name="TemperatureProvider">
  </definitions>
```

The WSDL contents: types

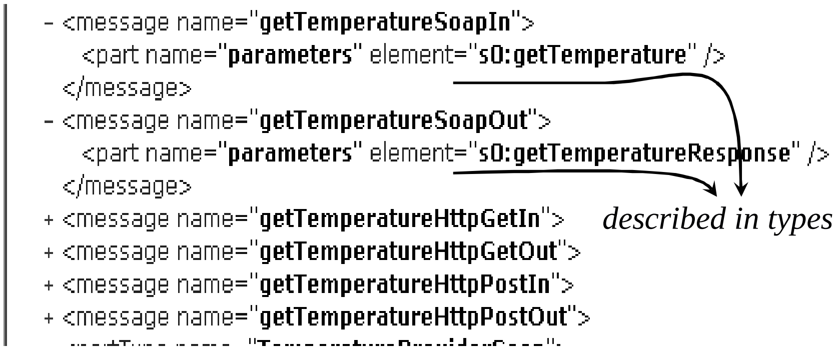
- Describes data types used in the document



```
- <types>
  - <s:schema elementFormDefault="qualified"
      targetNamespace="http://www.durasoftcorp.com/wsdl">
    - <s:element name="getTemperature">
      - <s:complexType>
        - <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="city" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    - <s:element name="getTemperatureResponse">
      - <s:complexType>
        - <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="getTemperatureResult"
              type="s:double" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="double" type="s:double" />
    </s:schema>
  </types>
```

The WSDL contents: message

- Describes what server accepts and returns

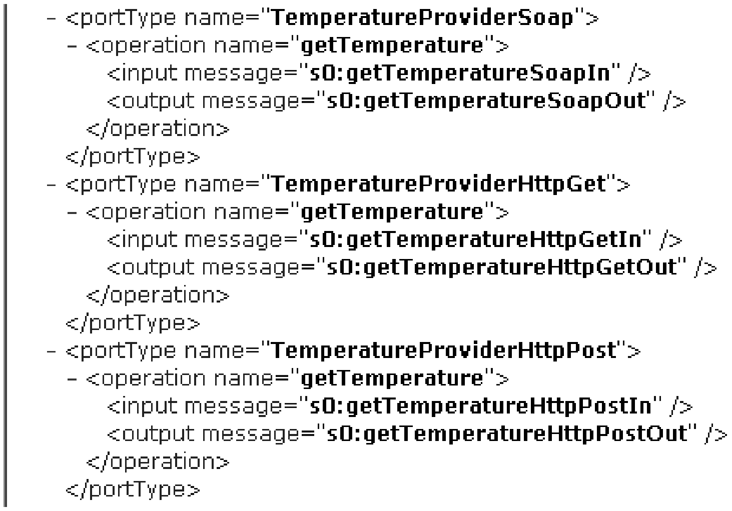


```
- <message name="getTemperatureSoapIn">
    <part name="parameters" element="s0:getTemperature" />
  </message>
- <message name="getTemperatureSoapOut">
    <part name="parameters" element="s0:getTemperatureResponse" />
  </message>
+ <message name="getTemperatureHttpGetIn">
+ <message name="getTemperatureHttpGetOut">
+ <message name="getTemperatureHttpPostIn">
+ <message name="getTemperatureHttpPostOut">
```

described in types

The WSDL contents: portType

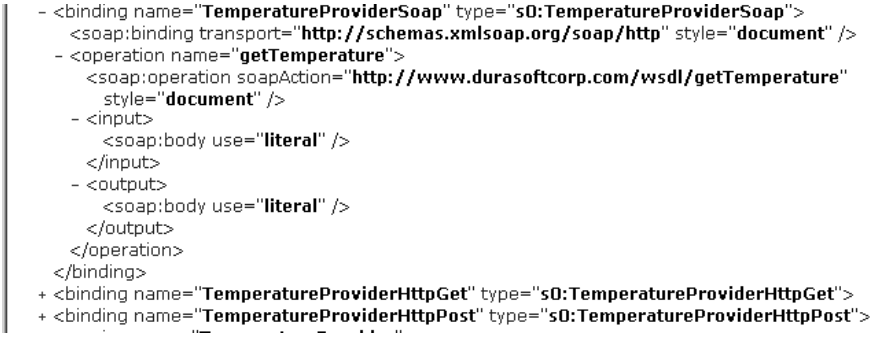
- Describes messages each operation accepts and returns



```
- <portType name="TemperatureProviderSoap">
  - <operation name="getTemperature">
      <input message="s0:getTemperatureSoapIn" />
      <output message="s0:getTemperatureSoapOut" />
    </operation>
  </portType>
- <portType name="TemperatureProviderHttpGet">
  - <operation name="getTemperature">
      <input message="s0:getTemperatureHttpGetIn" />
      <output message="s0:getTemperatureHttpGetOut" />
    </operation>
  </portType>
- <portType name="TemperatureProviderHttpPost">
  - <operation name="getTemperature">
      <input message="s0:getTemperatureHttpPostIn" />
      <output message="s0:getTemperatureHttpPostOut" />
    </operation>
  </portType>
```

The WSDL contents: binding

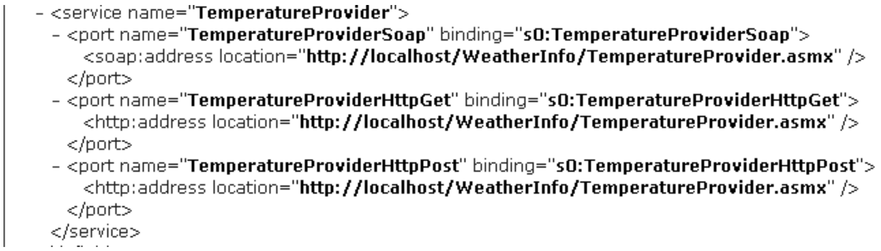
- Describes protocol used for each portType



```
- <binding name="TemperatureProviderSoap" type="s0:TemperatureProviderSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  - <operation name="getTemperature">
      <soap:operation soapAction="http://www.durasoftcorp.com/wsdl/getTemperature"
        style="document" />
    - <input>
        <soap:body use="literal" />
      </input>
    - <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
+ <binding name="TemperatureProviderHttpGet" type="s0:TemperatureProviderHttpGet">
+ <binding name="TemperatureProviderHttpPost" type="s0:TemperatureProviderHttpPost">
```

The WSDL contents: service

- Describes physical address and set of named ports for the web service



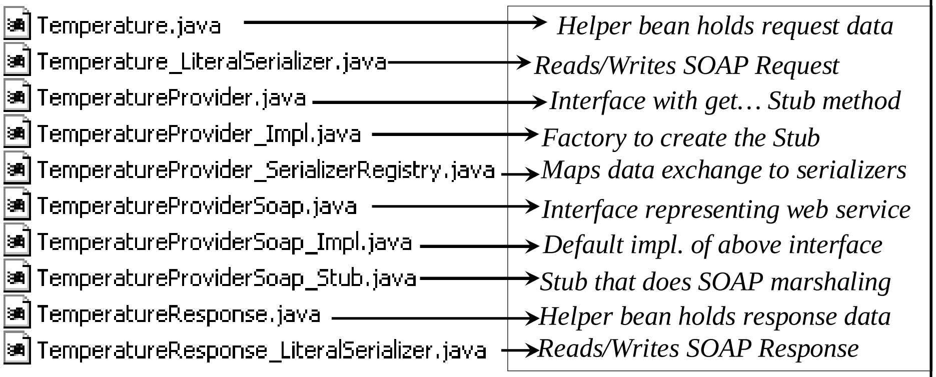
```
- <service name="TemperatureProvider">
  - <port name="TemperatureProviderSoap" binding="s0:TemperatureProviderSoap">
      <soap:address location="http://localhost/WeatherInfo/TemperatureProvider.asmx" />
    </port>
  - <port name="TemperatureProviderHttpGet" binding="s0:TemperatureProviderHttpGet">
      <http:address location="http://localhost/WeatherInfo/TemperatureProvider.asmx" />
    </port>
  - <port name="TemperatureProviderHttpPost" binding="s0:TemperatureProviderHttpPost">
      <http:address location="http://localhost/WeatherInfo/TemperatureProvider.asmx" />
    </port>
  </service>
```

Generating the Client Stub/Proxy

- Java Client Stub
  - xrpcc is a utility useful to generate a client stub for a web service from its WSDL
- .NET Client Proxy
  - wsdl is a utility useful to generate a client proxy for a web service from its WSDL
  - You may specify the language
    - default language is C#

---

xrpcc Generated Java Code



Temperature.java	Helper bean holds request data
Temperature_LiteralSerializer.java	Reads/Writes SOAP Request
TemperatureProvider.java	Interface with get… Stub method
TemperatureProvider_Impl.java	Factory to create the Stub
TemperatureProvider_SerializerRegistry.java	Maps data exchange to serializers
TemperatureProviderSoap.java	Interface representing web service
TemperatureProviderSoap_Impl.java	Default impl. of above interface
TemperatureProviderSoap_Stub.java	Stub that does SOAP marshaling
TemperatureResponse.java	Helper bean holds response data
TemperatureResponse_LiteralSerializer.java	Reads/Writes SOAP Response

wsdeploy Generated Client Stub

```
public class TemperatureProvider_Impl
      extends com.sun.xml.rpc.client.BasicService
      implements TemperatureProvider {
 …
 public TemperatureProviderSoap
      getTemperatureProviderSoap()
 { … return stub; }
}
```

```
public class TemperatureProviderSoap_Stub
    extends com.sun.xml.rpc.client.StubBase
    implements TemperatureProviderSoap {
 …
 public double getTemperature(java.lang.String city)
        throws java.rmi.RemoteException
 { // Generates SOAP Request, sends to server
   // Receives SOAP response, parses and returns result
 }
}
```

---

wsdl Generated .NET Client Proxy

```
namespace WeatherInformation {
…
public class TemperatureProvider :
      SoapHttpClientProtocol
{
      public TemperatureProvider() {
            this.Url =
            "http://localhost/WeatherInfo/TemperatureProvider.asmx";
      }
      public System.Double getTemperature(string city)
      {
        object[] results = this.Invoke("temperature",
                  new object[] {city});
        return ((System.Double)(results[0]));
      }
…
```

wsdl Generated .NET Client Proxy…

```
…
      public System.IAsyncResult BegingetTemperature(
            string city, System.AsyncCallback callback,
                  object asyncState) {

            return this.BeginInvoke(“getTemperature",
                  new object[] {city}, callback,
                        asyncState);
        }
      public System.Double
            EndgetTemperature(System.IAsyncResult
                         asyncResult) {
            object[] results =
                  this.EndInvoke(asyncResult);
            return ((System.Double)(results[0]));
        }
    }
}
```

Quiz Time!

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

Java Web Services Developer Pack 1.0_01

- JWSDP 1.0_01 contains the following:
- Java XML Pack
  - Java API for XML Messaging (JAXM 1.1_01)
    - API to send and receive SOAP messages via HTTP
  - SOAP with Attachments API for Java (SAAJ 1.1_02)
    - SOAP library – collection of classes (used by JAX-RPC and JAXM)
  - Java API for XML Processing (JAXP 1.2_01)
    - API for parsing – DOM, SAX parsing, XSLT, etc.
  - Java API for XML Registries (JAXR 1.0_02)
    - API to access business registries (UDDI)
  - Java API for XML-based RPC (JAX-RPC 1.0_01)
    - API to make SOAP calls from Java code without dealing with XML directly
- JSP Standard Tag Library (JSTL)
- Tomcat (Java Servlet & JSP container & tools)
- Ant build tool
- deploytool Web application deployment utility
- Registry server

JAX-RPC

- JAX-RPC
  - – maps WSDL ports to RMI interfaces
  - – Provides a RMI based Java interface to calling XML Web Services
- Following packages are part of JAX-RPC
  - – javax.xml.rpc.server
    - API for server based JAX-RPC end point model
  - – javax.xml.rpc
    - core JAX-RPC API for client programming model
  - – javax.xml.rpc.encoding
    - API to deal with serialization/encoding of messages
  - – javax.xml.rpc.handler
    - API for message handler
  - – javax.xml.rpc.handler.soap
    - API for SOAP message handler
  - – javax.xml.rpc.holders
    - Standard Java Holder classes (objects that hold various data types)
  - – javax.xml.rpc.soap
    - API for SOAP binding

---

javax.xml.rpc.server

- ServerLifeCycle interface
  - – defines lifecycle interface for service end point
  - – service provider will call the init and destroy to manage service life time
- ServletEndpointContext interface
  - – JAX_RPC runtime maintained endpoint context
  - – *ServerLifeCycle's init method sends this to the service*

ServletEndpointContext Interface

- Means for service to interact with the provider

```
public interface ServletEndpointContext
{
      public HttpSession getHttpSession();
      public MessageContext getMessageContext();
      public ServletContext getServletContext();
      public Principal getUserPrincipal();
}
```

---

javax.xml.rpc

- Interfaces
  - Call
    - Supports dynamic invocation of service end point
    - dynamic generation of stub instead of static stub usage
    - invoke method can take operation name and parameters
  - Service
    - Serves as factory for dynamic client side stub generation
  - Stub
    - base interface for the client side stub class
- Classes
  - NamespaceConstants
  - ParameterMode
    - enumeration that defines parameter passing mode – IN, OUT, INOUT
  - ServiceFactory
    - abstract factory for creation of implementations of Service interface

Stub Interface

- Base interface for client side stubs

```
public interface Stub
{
      public static String ENDPOINT_ADDRESS_PROPERTY;
      public static String PASSWORD_PROPERTY;
      public static String SESSION_MAINTAIN_PROPERTY;
      public static String USERNAME_PROPERTY;

      public Object _getProperty(String name);
      public Iterator _getPropertyNames();
      public void _setProperty(String name,
                                    Object value);
}
```

Stateless Nature

- Web Services are by nature stateless
- In Java Web Service, one object is created and shared among different clients
  - just like how a Servlet is
- Generally better not to maintain state
- Some times it becomes necessary though

Stateless Nature…

Service code to manage state

```
public void init(Object context) {
theContext =
 (javax.xml.rpc.server.ServletEndpointContext) context;
}
// In your web method
public String getInfo() {
  String str = "[First visit!]";
  //  We will check if this is the first visit
  javax.servlet.http.HttpSession session =
  theContext.getHttpSession();
  if (session.isNew())
            session.setAttribute("visit", "1");
  else  {
      int visitCount = Integer.parseInt(
        session.getAttribute("visit").toString());
…
}
```

Client code to manage state

```
stub._setProperty(
  javax.xml.rpc.Stub.SESSION_MAINTAIN_PROPERTY,
      new Boolean(true));
```

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

System.Web.Services namespace

- Namespace with classes to build web services and clients using ASP.NET and XML
- Fundamentally employs HTTP, XML, XSD, SOAP, and WSDL
- Consists of the following classes:
  - **WebService**
  - **WebMethodAttribute**
  - **WebServiceAttribute**
  - **WebServiceBindingAttribute**

WebService Class

- Your web service may derive from this
- Provides access to ASP.NET application and session object
- Generally, ***web services are stateless***
- If you like to keep state information, you may use the application or session object facilitated by this class
  - if you have no need for these, you do not have to derive from WebService class

WebMethodAttribute

- Used to mark the method as callable from remote web client
- Properties:
  - BufferResponse
    - response buffered (default) or not (not = 16Kb buffered)
  - CacheDuration
    - number of seconds response held in cache
  - Description
    - descriptive message for the XML web service method
  - EnableSession
    - session state will be enabled or not for the web method
  - MessageName
    - Name to be used for web method in the request / response
  - TransactionOption
    - indicates participation and desired options for transaction, like, requiring new transaction or transaction not supported, etc.

WebServiceAttribute

- Useful to control properties on the web service
- Properties:
  - Description
    - Descriptive message for the XML Web Service
  - Name
    - Name to be used for the web service
  - Namespace
    - Namespace to be used for the web service

WebServiceBindingAttribute

- Recollect the binding in WSDL?
- It specifies the protocol used for each operation
- You can specify a number of WebServiceBinding attributes
- Once specified, a web service method can use the default binding or a binding specified by one of these attributes
- Properties:
  - Location
    - location where the binding is defined
  - Name
    - name of the binding
  - Namespace
    - namespace associated with the binding

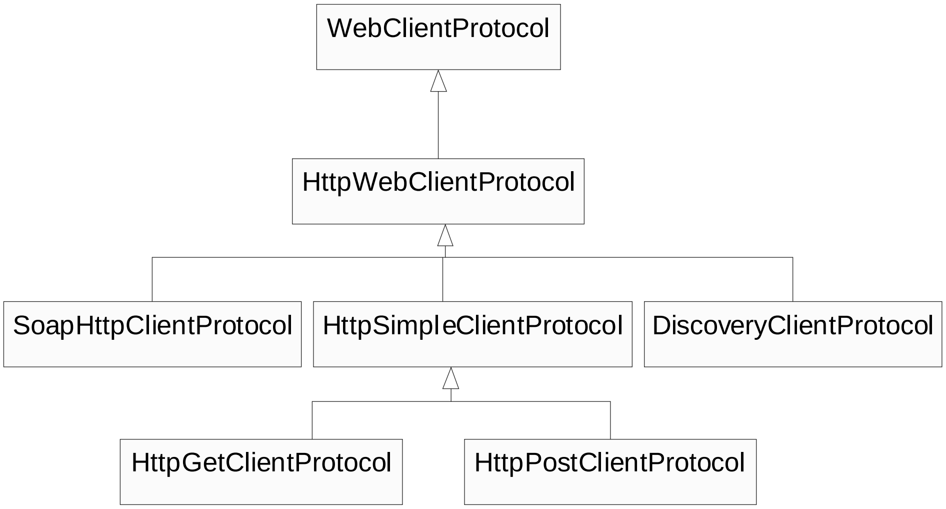
Other Namespaces

- System.Web.Services.Configuation
  - classes to configure the attributes related to SOAP extensions
- System.Web.Services.Description
  - Classes that help with generation of WSDL
- System.Web.Services.Discovery
  - classes that assist with Disco
- System.Web.Services.Protocols
  - classes that define the protocol to be used in transmission between Web Service and its client
  - Notice that your client proxy derived from the **SoapHttpClientProtocol** which is part of this namespace

WebClientProtocol

- This class is a base class from which
  - SoapHttpClientProtocol is indirectly derived for SOAP request
  - Controls the behavior of transport used to transmit XML Web Service request and response
  - To communicate with a Web Service, your proxy class derives directly or indirectly from WebClientProtocol

The Protocol Class Hierarchy



WebClientProtocol

HttpWebClientProtocol

SoapHttpClientProtocol

HttpSimpleClientProtocol

DiscoveryClientProtocol

HttpGetClientProtocol

HttpPostClientProtocol

Quiz Time!

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- ⇨ Synchronous vs. Asynchronous calls
- Conclusion

Is this really Asynchronous in .NET?

- Not really
- True asynchrony requires other protocols like SMTP, or the use of a MOM etc.
- .NET proxy kind of gives you an illusion of asynchrony
- It simply creates a thread for you in the back ground, waits for response, and invokes a call back when the response is received

A closer look at the Proxy class

- SoapHttpClientProtocol methods of interest
  - public Abort()
    - cancels a request to a web service method
  - protected Object[] Invoke(
    string methodName, Object[] parameters)
    - Invokes the XML Web Service synchronously
    - Generates the request XML, transmits & parses response
    - You would never invoke this method directly
      - derived proxy will have specific methods to invoke this
  - protected IAsyncResult BeginInvoke(
    string methodName, Object[] parameters,
    AsyncCallback callback, Object asyncState)
  - protected EndInvoke(IAsyncResult asyncResult)
    - useful for asynchronous web service method invocation

Making an Asynchronous call

- Your proxy has three special methods for each web service methods:
  - – yourWebServiceMethod [example: *temperature]*
  - – BeginyourWebServiceMethod [*Begintemperature*]
  - – EndyourWebServiceMethods [*Endtemperature*]
- You can
  - – invoke the Being… method
  - – go about doing other things you have to do
  - – finally, when ***you are ready***, call the End… method

---

Being Notified on Asynchronous call

- In the above example, you have to check back when you are ready for the result

- Would it be nice to be notified when result is ready?

- You can register a callback **delegate** for notification when the method is complete!

Overview of the Presentation

- State of distributed computing
- Application of XML
- SOAP
- An example Java Web Service and Clients
- An example .NET Web Service and Clients
- WSDL and related tools
- Java WS Classes and Proxy
- .NET WS Classes and Proxy
- Synchronous vs. Asynchronous calls
- Conclusion

Impact of Web Services

- Web Services will change the way we develop systems
- It provides true interoperability between systems
  - independent of languages
  - independent of processors
  - independent of platforms
- Next generation of components will provide true reusability
- Definitely an exciting time

Integration of JWS and .NET WS

- Server side issues
- Writing Service
  - Java Web Service
    - Seven step process
    - Mostly copy and paste deployment descriptors
  - .NET Web Service
    - Two step process
    - Optionally may use Visual Studio .NET
- Managing State
  - Pretty much the same process in both

Integration of JWS and .NET WS…

- Client side issues
- Writing Client
  - Java Client
    - Four step process
  - .NET Client
    - Three step process
- Managing State
  - pretty much the same process in both
- Setting the target port
  - Need to set it for Java Client Stubs
  - No need to set it for .NET client proxies
    - Note, you need to set the stub’s Url for .NET client if using port other than 80 – bug

What do we need to learn?

- In order to ***develop*** with Web Services, here is what we need:
  - Knowledge of XML
  - Knowledge of XML Schema is very useful
  - SOAP architecture
  - WSDL
  - Web Services Programming API
    - .NET or Java based

References

- http://www.w3.org (Web Services, SOAP)
- http://java.sun.com (Java Web Services)
- http://msdn.microsoft.com (.NET Web Services)
- http://www.agiledeveloper.com/download.aspx

Please fill in the evaluation & turn it in to the registration desk