SOAP Web Services Creation& Invocation

Laboratory of Service Design and Engineering

2011/2012

Outline

- What is SOAP?
- Web Services with SOAP
- WSDL
- Client Implementation
- Exercise

Simple Object Access Protocol

What is SOAP?

1. SOAP stands for Simple Object Access Protocol
2. SOAP is essentially an XML-based protocol for invoking remote methods.
3. SOAP is a protocol for accessing a Web Service.
4. SOAP is a format for sending messages
5. SOAP communicates via Internet
6. SOAP is platform independent
7. SOAP is language independent
8. SOAP is based on XML

Simple Object Access Protocol

- A SOAP message is an ordinary XML document containing the following elements:
  - An Envelope element that identifies the XML document as a SOAP message
  - A Header element that contains header information
  - A Body element that contains call and response information
  - A Fault element containing errors and status information

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

A SOAP Request Example

SOAP Request

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

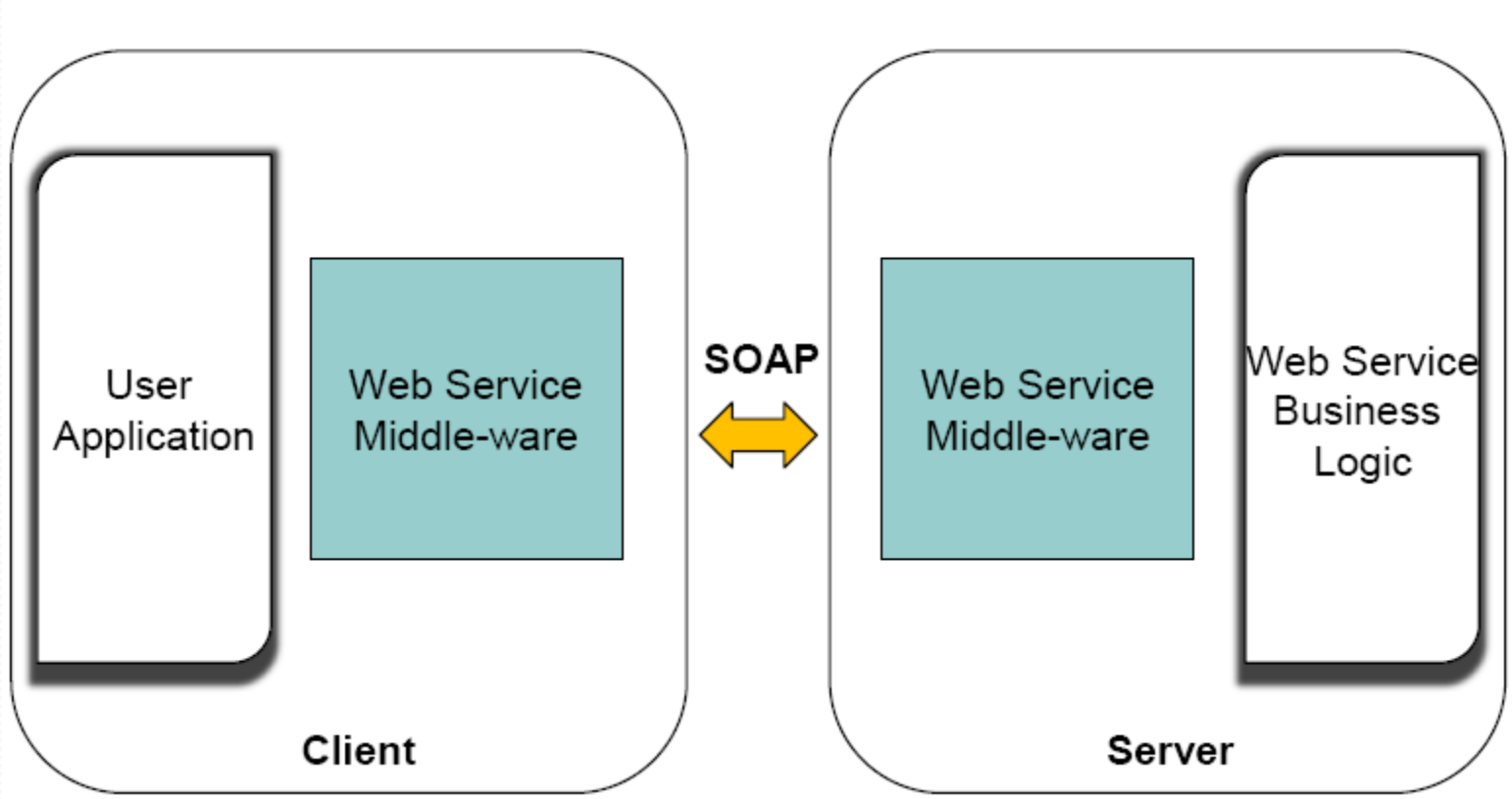
SOAP Response Example

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Web Service Middle-Ware



Invoking JWS Service

- Two options
  - Using Dynamic Invocation Interface (DII)
  - Using Stubs generated from service WSDL description

Client Code: DII Method

```
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import javax.xml.namespace.QName;

public class TestAddFunction {
  public static void main(String[] args) {
    try {
       String endpoint = "http://localhost:8080/axis/AddFunction.jws";
       Service service = new Service();
       Call call = (Call) service.createCall();
       call.setOperationName(new QName(endpoint, "addInt"));
       call.setTargetEndpointAddress(new java.net.URL(endpoint));
       Integer ret = (Integer) call.invoke(new Object[] { new
Integer(5), new Integer(6) });
       System.out.println("addInt(5, 6) = " + ret);
    } catch (Exception e) {
       System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

Web Service Description Language

WSDL

- WSDL stands for Web Services Description Language.
- WSDL is a document written in XML.
- The document describes a Web service.
- It specifies the location of the service and the operations (or methods) the service exposes.

WSDL

- A WSDL document describes a web service using these major elements:
- **<types>**
  - The data types used by the web service
- **<message>**
  - The messages used by the web service
- **<portType>**
  - The operations performed by the web service
- **<binding>**
  - The communication protocols used by the web service

WSDL Example

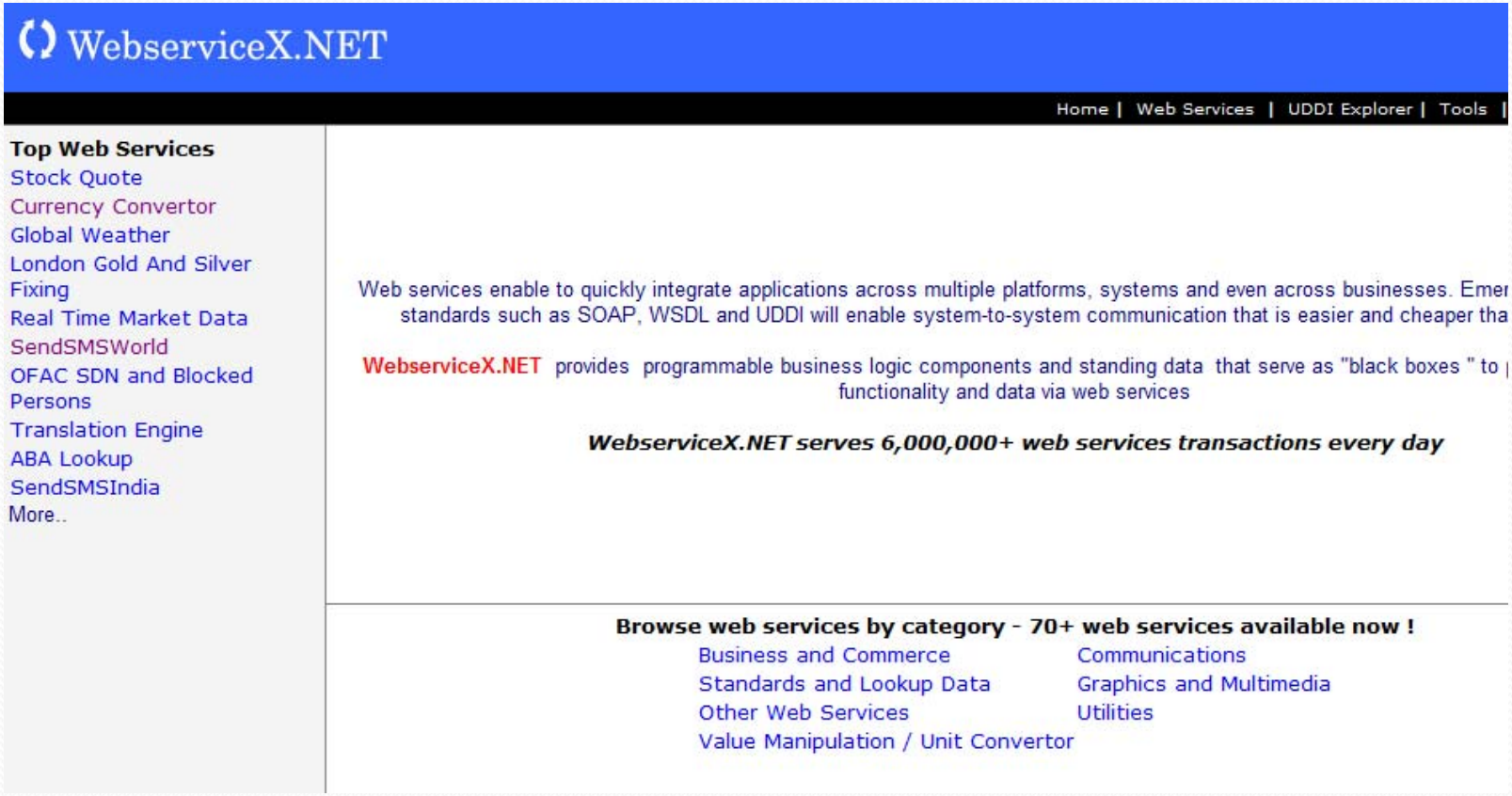
```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Example Service

- Length Convertor Service



Length Convertor

- What we need
  - A service (Already implemented at webservicex.net)
  - Service WSDL
    - Download it http://www.webservicex.net/length.asmx?WSDL
  - Stubs
    - Generate stubs from WSDL
  - Write client using stubs
  - Access the service (pass parameter and get results)

Lets do it

Exercise

- Access the Currency Convertor Service
  - Download WSDL file from
  - http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
- Generate stubs using WSDL file.
  - Use given build.xml
- Create Client using stubs.
- Convert any currency to other currency type.

Create your own service

- Create your own service
  - Implement your service logic
  - Define one service class which exposes service methods
- Compile and deploy it to apache/axis folder
- Generate WSDL from your service
- Use that WSDL to write client program

Customer Order Service

```
public class Order {

    String productName;
    int quantity;

    public Order() {
        // TODO Auto-generated constructor stub
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
```

Order Service

```
public class Client {

    String name;
    int age;
    String adrs;
    Order [] orders;

    public Client() {
        // TODO Auto-generated constructor stub
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
```

Service Class

```
public class OrderService {


    public String placeOrder(Client client)
    {
        return client.getName() + " Your order has been placed.";
    }

}
```

deploy.wsdd

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="ClientOrderService" provider="java:RPC">
  <parameter name="className" value="soaa.service.OrderService"/>
  <parameter name="allowedMethods" value="*"/>


  </service>



  <beanMapping qname="ns:Client" xmlns:ns="urn:ClientNS" languageSpecificType="java:soa
  <beanMapping qname="ns:Order" xmlns:ns="urn:OrderNS" languageSpecificType="java:soaa.

</deployment>
```

Undeploy.wsdd

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/"
              xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="ClientOrderService" provider="java:RPC">
  <parameter name="className" value="soaa.service.OrderService"/>
  <parameter name="allowedMethods" value="*"/>


  </service>


</undeployment>
```

Exercise

- Make your own customer service
- Deploy this service
- Write your client to access it (as we did in last lab)
- Call the service