Homework #2
Web Service Programming
ID2208

SOAP & WSDL

KTH ICT School
VT 2010

This presentation is based on a tutorial published by IBM and some SUN Micorosystems (currently Oracle) documents pointed out in the reference.

Aims

To learn the followings:

- WSDL and SOAP

- Design and Developing Web Services

- Developing Web Service Client

- SOAP processing, SOAP Attachment

Homework #2- Part 1

Developing Web Service and Web Service Client

Caution !

In THEORY you could use any application server, any WS libraray with any IDE for developing Web Service technologies ,

But in PRACTICE you might face some strange problmes (due to incompatbility of versions, non standarad libraries,....) .

So BE READY to face some headaches, just take your time and look for possible solutions.We have tested the recommended tools and libraries ,and hopefully you will get less headaches.

Java API for XML Web Services (JAX-WS)

●JAX-WS is designed to simplify building Web Services in Java (using Annotations,etc)

●Part of Java SE 6 and Java EE 6 platforms

●It follows annotated based programming model

It also includes :

• SAAJ (SOAP with Attachments API for Java)

  • provides a standard way to deal with SOAP messages with XML attachment

●JAX-WS Architecture:

https://jax-ws-architecture-document.dev.java.net/nonav/doc/?jaxws/package-summary.html

Current Version is JAX-WS 2.0/2.1

Libraries & Tools Installation -1

1. Download and Install JWSDP 2.0
Alternatively you can download latest release of JAX-WS RI 2.1.5 (reference implementation) from:
https://jax-ws.dev.java.net/2.1.5/
(JAX-WS RI 2.1.5 comes with more samples !)

2. Download and install an Application Server
Since it is mentioned explicitly in the documents that "*All of JAX-WS samples are tested to run on Glassfish v2 UR2 build and on Apache Tomcat 5.x* ", lets choose only these two versions:

- GlassFish: https://glassfish.dev.java.net/downloads/v2ur2-b04.html
- Apache Tomcat : http://tomcat.apache.org/download-55.cgi

6

Libraries & Tools Installation -2

3. You may want to use IDEs as a development environment for web services and take advantage of its features for easy development:

-NetBeans 6.8: http://www.netbeans.org/
If you choose "All" version it also comes with GalssFish and Tomcat.

-Eclipse 3.5 : http://www.eclipse.org/downloads/
-Eclipse WPT plug-in: http://www.eclipse.org/webtools/

Eclipse has plug-ins for GlassFish https://glassfishplugins.dev.java.net/
and used to have a plug-in for Apache Tomcat .

* **We recommend** : NetBeans + GlassFish but NetBeans + Tomcat more likely will be fine too .

Or even better way, use no IDE, just " vi" or "Notepad" !

Two Ways to Develop Web Service

1-*Top-Down approach*: Start with a WSDL contract, and generate the service implementing Java class .It requires good understanding of WSDL and XSD *(WSDL to JAVA).*

2-*Bottom-Up approach*: Start with a developing Java class, and use annotations to generate both a WSDL file and a Java interface. Good starting point If you are new to Web Service *(JAVA to WSDL)*.

For the time being, we follow Bottom-Up approach.

Homework #2- JAX-WS

APIs available to the developer in the Java SE 6 platform (from SUN JWSDP documents).

API	*Package*
JAX-WS	javax.xml.ws
SAAJ	javax.xml.soap
Web-service meta-data	javax.xml.jws

Steps for Creating the Web Service and Client

1-Code the service implementation class.

2-Compile the implementation class. Use *wsgen* to generate the artifacts required to deploy the service.

3-Package the files into a WAR file. Deploy the WAR file. The web service artifacts (which are used to communicate with clients) are generated by the Application Server during deployment.

4-Code the client class. Use *wsimport* to generate and compile the web service artifacts needed to connect to the service.

5-Compile the client class and Run the client.

Warm up Exercise – OrderProcessing Service

- Follow tutorial provided by IBM:

  https://www6.software.ibm.com/developerworks/education/ws-jax/ws-jax-a4.pdf

- Download the sample code referenced at the end of the document

- Do the tutorial before starting the Homework

OrderProcessing Service

An Order-Processing Web service that accepts order information, shipping information, and ordered items, and ultimately generates a confirmation ID as a response.

Developing Server Side Web Service - 1

First write the service code:

1- Write a service implementation in Plain Old Java Object.

2- Add JWS Annoattion tags (JSR-181)

- Add One ”@WebService” tag per class. It gives information like TargetNamespace, Port Name and Service Name

   - Optionally add "@WebMethod" tag per each public methods to make them Web Service Operations or don't tag any of them.

   - Optionally, add ”@SOAPBinding” tag per class. It specifies the bindings to generate the WSDL file

13

@WebService annotation

Class	Name	Description
String	endpointInterface	The complete name of the service endpoint interface (SEI) defining the service's abstract web service contract.
String	name	The web service's name.
String	portName	The web service's port name.
String	serviceName	The web service's service name.
String	targetNamespace	If the @WebService.targetNamespace annotation is on an SEI, the targetNamespace is used for thenamespace for the wsdl:portType and associated XML elements.
String	wsdlLocation	The location of a predefined WSDL file describing the service.

(from SUN Micorsystems documents)

Useful hints on JAX-WS annotation:

https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html#2.6 javax.jws.HandlerChain|outline

@WebService annotation

```java
@WebService (name = "AddNumbers",

    targetNamespace = "http://duke.org")
public class AddNumbersImpl {
/**
 * @param number1
 * @param number2
 * @return The sum
 * @throws AddNumbersException
 * if any of the numbers to be added is negative.
 */
public int addNumbers(int number1, int number2) throws AddNumbersException
{

    if (number1 < 0 || number2 < 0) {
      throw new AddNumbersException("Negative number cant be added!",
      "Numbers: " + number1 + ", " + number2);
      }
    return number1 + number2;
    }
}
```

Order Processing Web Service Implementation
Plain Java class

```java
public class OrderProcessService
 {
public OrderBean processOrder(OrderBean orderBean) {
// Do processing...
System.out.println("processOrder called for customer"+
orderBean.getCustomer().getCustomerId());
// Items ordered are
if (orderBean.getOrderItems() != null) {
System.out.println("Number of items is " +
orderBean.getOrderItems().length);
    }
//Process Order.
//Set the Order Id.
orderBean.setOrderId("A2234");
    return orderBean;
}
```

OrderBean class

```
public class OrderBean {

private Customer customer;
private Address shippingAddress;
private OrderItem[] orderItems;
private String orderId;

public Customer getCustomer() {
    return customer;
}
public void setCustomer(Customer customer) {
    this.customer = customer;
}
public String getOrderId() {
    return orderId;
}
public void setOrderId(String orderId) {
    this.orderId = orderId;
}
public Address getShippingAddress() {
    return shippingAddress;
}
```

Order Processing Service – Add Annotation

```
@WebService(serviceName = "OrderProcess",
    portName = "OrderProcessPort",
    targetNamespace = "http://jawxs.ibm.tutorial/jaxws/orderprocess")

@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,use=SOAPBinding.Use.LITERAL,par
ameterStyle=SOAPBinding.ParameterStyle.WRAPPED)

public class OrderProcessService {
@WebMethod
public OrderBean processOrder(OrderBean orderBean) {
    // Do processing...
        System.out.println("processOrder called for customer"+
                                orderBean.getCustomer().getCustomerId());

    // Items ordered are
    if (orderBean.getOrderItems() != null) {
        System.out.println("Number of items is "  + orderBean.getOrderItems().length);
    }
    //Process Order.
    //Set the Order Id.
    orderBean.setOrderId("A2234");
    return orderBean;}
```

Publishing the Web Service

Publish the Service:

1- Generate JAX-WS artifacts using "wsgen" tool.
wsgen -cp . com.ibm.jazws.tutorial.service OrderProcessService -wsdl

Generated Artifcats: The WSDL and Schema

2- Publish the service on a light weight web server
Endpoint.publish("http://localhost:8080/OrderProcessWeb/orderprocess",new OrderProcessService());

You should be able to see the WSDL at
http://localhost:8080/OrderProcessWeb/orderprocess?wsdl

Developing Web Service Client - 1

Create web service client (stubs) from WSDL using
"wsimport" .

app_server_root/bin/wsimport.sh [options] -wsdlLocation URI

Use the -verbose option to see a list of generated files when you run the command.

Use the -keep option to keep generated Java files.

Use the -wsdlLocation option to specify the location of the WSDL file.

wsimport -keep -p com.ibm.jaxws.tutorial.service.client
http://localhost:8080/OrderProcessWeb/orderprocess?wsdl

The generated artifacts will be placed at:

com.ibm.jaxws.tutorial.service.client

You don't need to change anything in those generated artifacts (classes). Just use them!

Developing Web Service Client - 2

- Artifacts ("wsimport" generated files) :
  - Service Endpoint Interface: *it contains the interface which your service implements* (OrderProcessService)

  - Service class (OrderProcess) . The service stub class which the client uses to make requests to service.

  - JAXB-generated class (Address, Cusomer, OrderBean, ...)

  - Exception class mapped from WSDL (we don't have)

  - Request and Response "Wrapper" classes (ProcessOrder, ProcessOrderResponse)

OrderProcessService (EndPoint Interface)

```
@WebService(name = "OrderProcessService", targetNamespace =
"http://jawxs.ibm.tutorial/jaxws/orderprocess")
@XmlSeeAlso({ ObjectFactory.class})
public interface OrderProcessService {
   /** @param arg0
    * @return  returns com.ibm.jaxws.tutorial.service.client.OrderBean */

   @WebMethod
   @WebResult (targetNamespace = "")
   @RequestWrapper(localName = "processOrder", targetNamespace =
"http://jawxs.ibm.tutorial/jaxws/orderprocess", className =
"com.ibm.jaxws.tutorial.service.client.ProcessOrder") <------ wrapper

   @ResponseWrapper(localName = "processOrderResponse", targetNamespace =
"http://jawxs.ibm.tutorial/jaxws/orderprocess", className =
"com.ibm.jaxws.tutorial.service.client.ProcessOrderResponse") <------ wrapper

   public OrderBean processOrder( @WebParam(name = "arg0", targetNamespace = "")
      OrderBean arg0);
}
```

OrderProcess (stub)

```java
@WebServiceClient(name = "OrderProcess", targetNamespace =
"http://jawxs.ibm.tutorial/jaxws/orderprocess", wsdlLocation =
"http://localhost:8080/OrderProcessWeb/orderprocess?wsdl")
public class OrderProcess   extends Service{
  private final static URL ORDERPROCESS_WSDL_LOCATION;
  static {
     URL  url = new URL("http://localhost:8080/OrderProcessWeb/orderprocess?wsdl");
     ORDERPROCESS_WSDL_LOCATION = url;
  }
  public OrderProcess(URL wsdlLocation, QName serviceName) {
     super(wsdlLocation, serviceName);  }

  public OrderProcess() {
     super(ORDERPROCESS_WSDL_LOCATION, new
QName("http://jawxs.ibm.tutorial/jaxws/orderprocess", "OrderProcess")); }
   /*    returns OrderProcessService*/

   @WebEndpoint(name = "OrderProcessPort")
   public OrderProcessService getOrderProcessPort() {    <----------------------
     return (OrderProcessService)super.getPort(new
QName("http://jawxs.ibm.tutorial/jaxws/orderprocess", "OrderProcessPort"),
OrderProcessService.class);
   }
```

23

Service Client Code

```
     .....
// WSDL url of the OrderProcess web services
 URL url =
getWSDLURL("http://localhost:8080/OrderProcessWeb/orderprocess?wsdl");
// Qualified name of the service
   QName qName = new QName(
        "http://jawxs.ibm.tutorial/jaxws/orderprocess", "OrderProcess");
// Call the stub class
OrderProcess orderProcessingService = new OrderProcess(url, qName);

   //  initialize the request object
OrderBean order = populateOrder();
  // get the proxy to the service. The proxy implements the service interface defined
by the service
OrderProcessService port = orderProcessingService.getOrderProcessPort();
// invoke the proxy by passing the OrderBean intsance
// and get the response in form of OrderResponse object
OrderBean orderResponse = port.processOrder(order);

System.out.println("Order id is " + orderResponse.getOrderId());
.....
```

Run Service Client

Run client class:

java *com.ibm.jaxws.tutorial.service.client.***OrderClient**

http://localhost:8080/OrderProcessWeb/orderprocess?wsdl

Output:

Order id is A1234

References:

Design and Develop JAX-WS 2.0 web services, IBM corporation

https://www6.software.ibm.com/developerworks/education/ws-jax/ws-jax-a4.pdf

From WSDL to Java?

1. Generate a service endpoint interface (using wsimport)

2. Implement the service endpoint interface.

3. Create a WAR file (deployment descriptor web.xml , sun-jaxws.xml , WSDL file and Service Impl. Class) to deploy.

4. Publish the End-point

5. Develop Client side as before

Look at "fromwsdl" example in .../jaxws/samples/

Homework #2 – Part 2

SOAP

SAAJ (SOAP Attachment API for Java)

-Is used to write SOAP messaging applications directly rather than use JAX-WS.

-By simply making method calls using the SAAJ API, you can read and write SOAP-based XML messages.

- SAAJ is used in SOAP Message handlers for reading and modifying SOAP headers

- This tutorial and most of the related materials are based on SAAJ section in SUN WSDP tutorial

http://java.sun.com/webservices/docs/2.0/tutorial/doc/

SOAP Message with no Attachment

Source: JAVA WSDP Tutorial
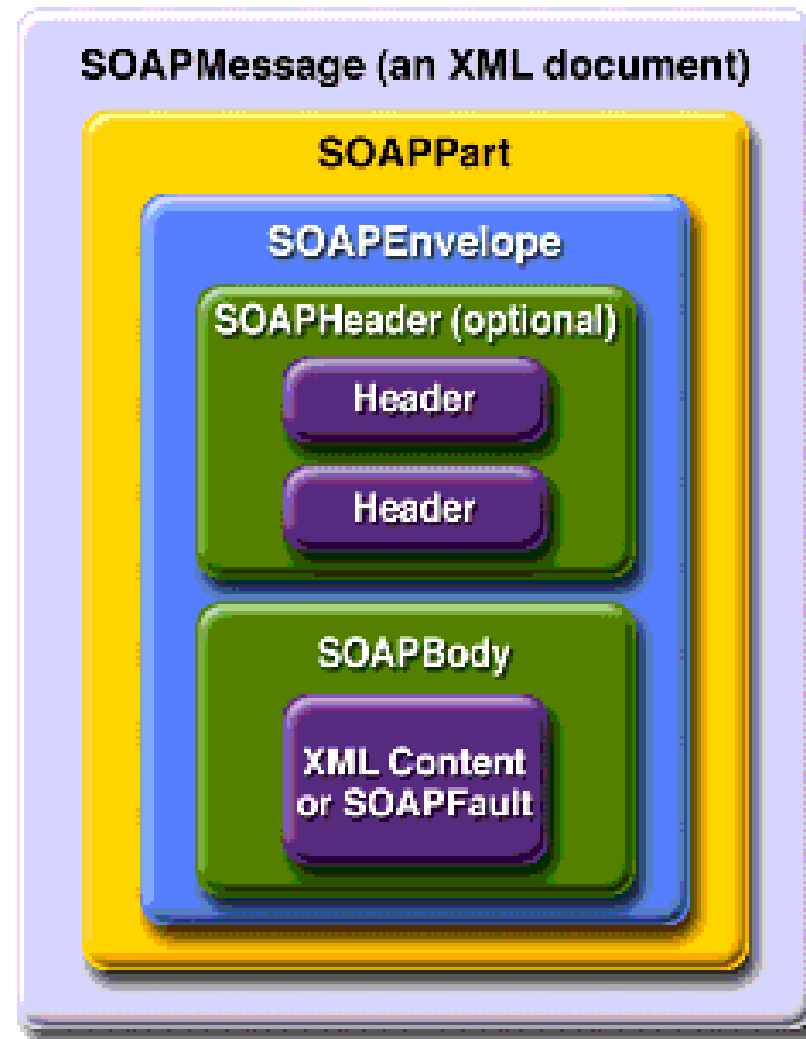
Building SOAP Message

1- Create (Get) SOAP message Part, Body and Header

2- Add element to Header (optional)

3- Add elements (e.g. request, response) to Body

4- Add content (e.g. an XML document) to Body

5- Add Attachments (optional)

6- Add Attributes (optional)

Steps to Create a SOAP Message -1

//get insance of SOAP factory
MessageFactory factory = MessageFactory.newInstance();

// get empty soap message
SOAPMessage message = factory.createMessage();

// get Part piece of the message
SOAPPart soapPart = message.getSOAPPart();

// get Envelop piece of the message
SOAPEnvelope envelope = soapPart.getEnvelope();

// access the empty header of the message
SOAPHeader header = envelope.getHeader();

// access the empty body of the message
SOAPBody body = envelope.getBody();

Steps to Create a SOAP message -2

What we have built so far:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">

    <SOAP-ENV:Header/>

 <SOAP-ENV:Body>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Steps to Create a SOAP message - 3

Adding content to Body :

```
SOAPBody body = message.getSOAPBody();
QName bodyName = new QName("http://wombat.ztrade.com",
  "GetLastTradePrice", "m");
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
QName name = new QName("symbol");
SOAPElement symbol = bodyElement.addChildElement(name);
symbol.addTextNode("SUNW");
```

We will have :

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="http://wombat.ztrade.com">
      <symbol>SUNW</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Send a SOAP Message

Get a SOAPConnection Object :

*SOAPConnectionFactory soapConnectionFactory =
SOAPConnectionFactory.newInstance();
SOAPConnection connection =
soapConnectionFactory.createConnection();*

Send SOAP Message *:*
*// Create an endpint point which is a URL to the published web service
java.net.URL endpoint =new URL("http://wombat.ztrade.com/quotes");*

*// Send the SOAP Message request and then wait for SOAP Message
response*

SOAPMessage response = connection.call(message, endpoint);

Read SOAP Message

SOAPBody soapBody = response.getSOAPBody();

Iterator iterator =   soapBody.getChildElements(bodyName);

SOAPBodyElement bodyElement = (SOAPBodyElement)iterator.next();
String lastPrice = bodyElement.getValue();

System.out.print("The last price for SUNW is ");
System.out.println(lastPrice);

```
<SOAP-ENV:Body>
      <elementName> 11.11 </elementName>
   ....
</SOAP-ENV:Body>
```

Adding Content to Header

```
SOAPHeader header = message.getSOAPHeader();

QName headerName = new Qname("http://ws-
org/schemas/conformanceClaim/", "Claim", "wsi");

SOAPHeaderElement headerElement
=    header.addHeaderElement(headerName);

headerElement.addAttribute(new
QName("conformsTo"),"http://ws-i.org/profiles/basic/1.1/");
```

We will have:

<SOAP-ENV:Header>
    <wsi:Claim xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
        conformsTo="http://ws-i.org/profiles/basic/1.1/"/>
</SOAP-ENV:Header>

36

Adding Content to Empty SOAP Body-1

```
SOAPBody body = message.getSOAPBody();
QName bodyName = new QName("http://sonata.fruitsgalore.com",
    "PurchaseLineItems", "PO");


SOAPBodyElement purchaseLineItems =  body.addBodyElement(bodyName);
QName childName = new Qname("Order");
SOAPElement order =  purchaseLineItems.addChildElement(childName);
childName = new Qname("Product");


SOAPElement product = order.addChildElement(childName);
product.addTextNode("Apple");
childName = new QName("Price");
SOAPElement price = order.addChildElement(childName);
price.addTextNode("1.56");
childName = new QName("Order");
SOAPElement order2 =  purchaseLineItems.addChildElement(childName);
childName = new QName("Product");
SOAPElement product2 = order2.addChildElement(childName);
product2.addTextNode("Peach");
childName = soapFactory.new QName("Price");
SOAPElement price2 = order2.addChildElement(childName);
price2.addTextNode("1.48");
```

37

Adding Content to Empty SOAP Body-2

The created XML fragment of body content:

```
<PO:PurchaseLineItems
 xmlns:PO="http://sonata.fruitsgalore.com">
  <Order>
    <Product>Apple</Product>
    <Price>1.56</Price>
  </Order>
  <Order>
    <Product>Peach</Product>
    <Price>1.48</Price>
  </Order>
</PO:PurchaseLineItems>
```

Adding XML document to SOAP Body as a DOM object

```
DocumentBuilderFactory dfactory = DocumentBuilderFactory.newInstance();

dfactory.setNamespaceAware(true);

dfactory.setValidating(true);

DocumentBuilder builder = dfactory.newDocumentBuilder();

Document document = builder.parse("CV.xml");

SOAPBodyElement docElement = body.addDocument(document);
```

SOAP Message with Attachment
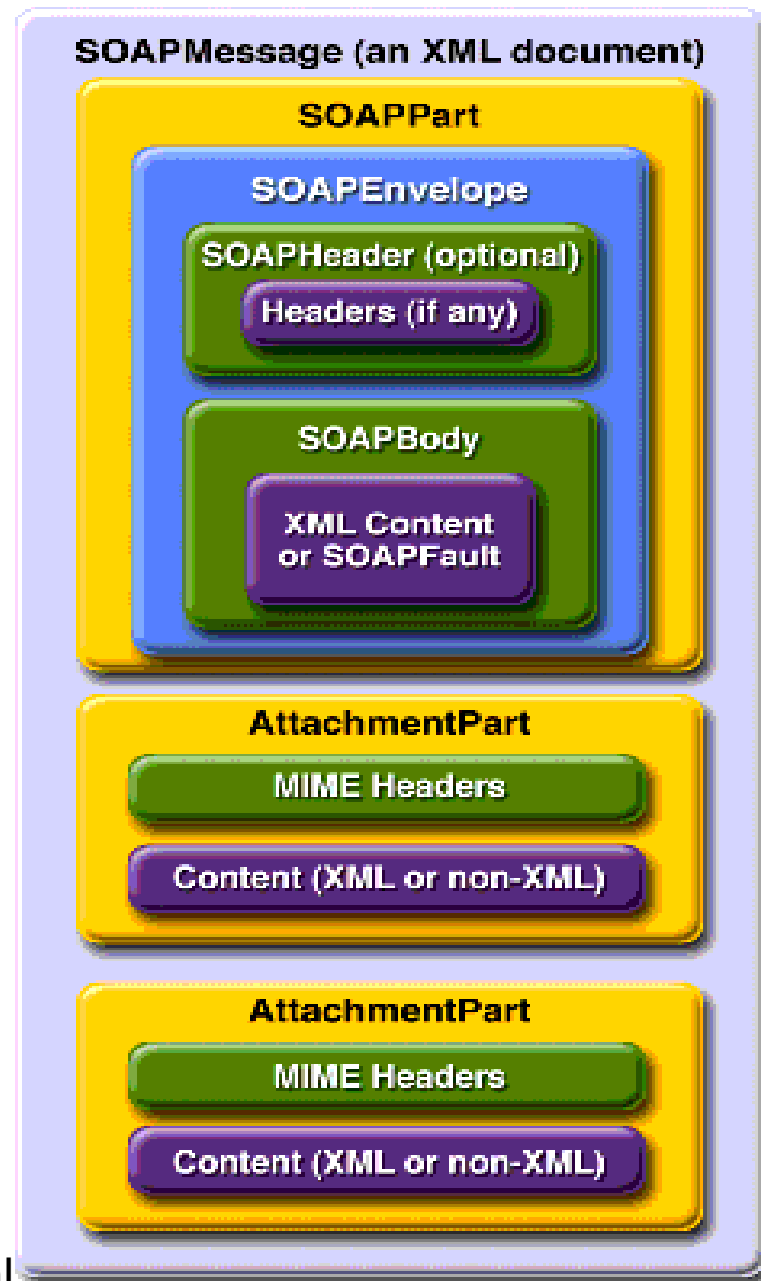
Adding Attachment to SOAP Message

//Adding Image file to SOAP Attachment

```
   URL url = new URL
("file:///home/shahab/Working/NetBeansProjects/Homework2_Clien/SOA.jpg");

 DataHandler dataHandler = new DataHandler(url);

        AttachmentPart attachment = message.createAttachmentPart(dataHandler);

        attachment.setContentId("SOA_image");

        message.addAttachmentPart(attachment);
```

There are also other ways to add XML or any file to SOAP Message. Look at the referenced link for SUN tutorial.

Handling SOAP Messages

SOAP message handlers are used to enable Web Services and their clients to access the SOAP message for additional processing (like processing SOAP header) of the message request or response.

Two kinds of SOAP message handlers:

1- SOAP handlers : capable to access the entire SOAP message, including the message headers and body.

2-Logical handlers: gives access to only the payload (eg. Body) of the message . For instance it can not change the protocol-specific information (like headers) in a message.

SOAP Message Handler
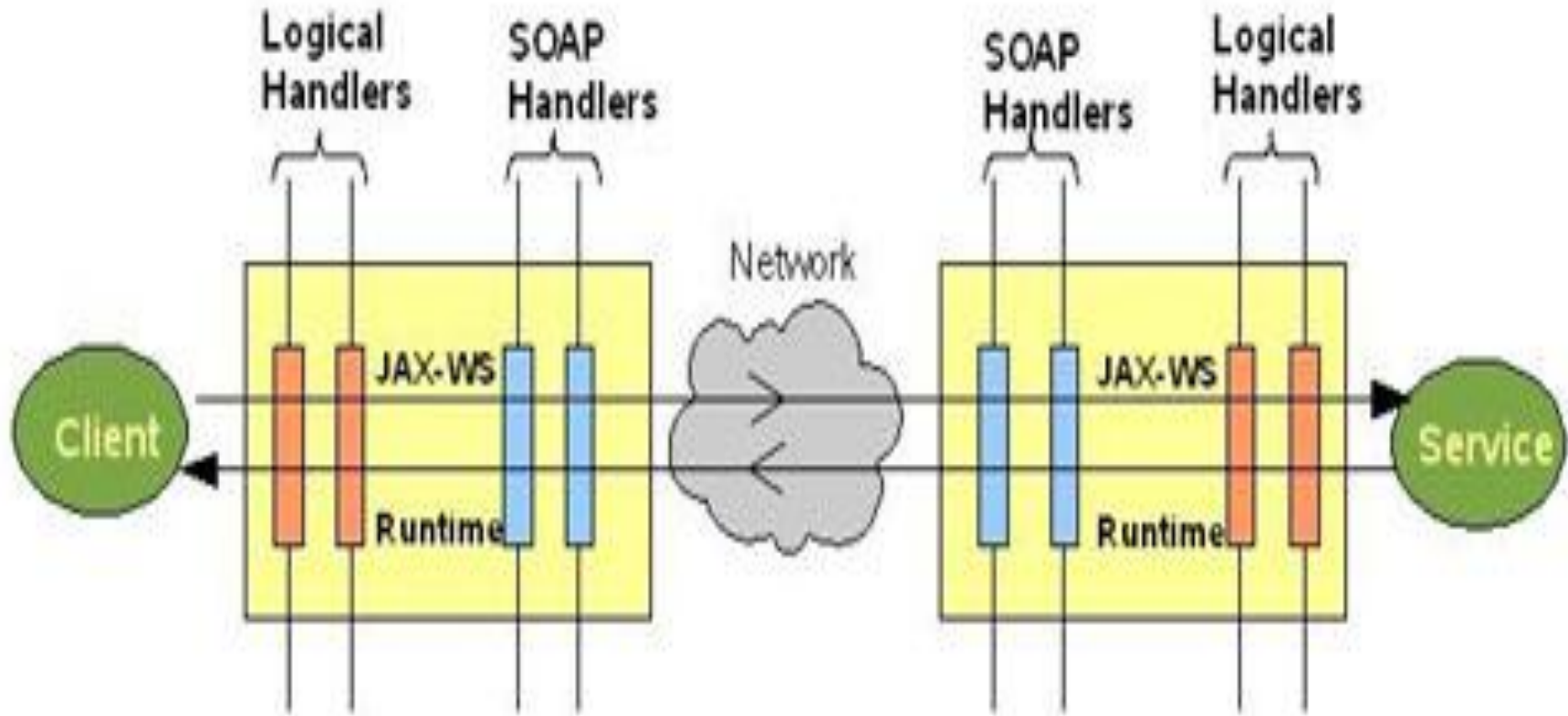
Picture form SUN Web Service tutorials

Steps to add SOAP Message Handler on Server side -1

1. Create SOAP Message handler class (say Handler1) by implementing "SOAPHandler<SOAPMessageContext>" interface.

2. Handling SOAP messages is done in "*handleMessage(SOAPMessageContext context)*" method.

public boolean handleMessage(SOAPMessageContext messageContext) {

    SOAPMessage soapMessage = messageContext.getMessage();

    SOAPPart soapPart = soapMessage.getSOAPPart();

    SOAPEnvelope soapEnvelope = soapPart.getEnvelope();

    SOAPBody soapBody = soapEnvelope.getBody();

    SOAPHeader soapHeader = soapEnvelope.getHeader();

    .....

Steps to add SOAP Message Handler on Server side -2

2. Create a handler configuration file say 'handlers.xml' specifying the chain of handlers intercepting the soap messages. In fact, it adds the following configuration into websevices.xml (service deployment description)

<?xml version="1.0" encoding="UTF-8"?>

<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">

  <handler-chain>

   <handler>

    <handler-class>examples.webservices.handler.Handler1</handler-class>

   </handler>

  </handler-chain>

Steps to add SOAP Message Handler on Server side -3

3.Put "@HandlerChain "annotation to instruct the JAX-WS runtime to apply the handlers configured in the above-mentioned configuration on to the web service Impl.

@WebService()

@HandlerChain(file = "TranscriptWS_handler.xml")

public class TranscriptWS {

   @WebMethod(operationName = "getTranscript")

   public String getTranscript(@WebParam(name = "StudentName")

4. Deploy and Run

soapUI

- soapUI is a free and open source application for monitoring, invoking, developing, simulating and functional testing of web services over HTTP.

- More on soapUI on http://www.soapui.org .

- It provides plug-ins for bith NetBean and Eclipse

- SoapUI plug-in for NetBeans:
  http://www.javapassion.com/handsonlabs/wssoapuibasics/

soapUI

You want more samples on JAX-WS?

Look at the "samples" directory in the "JAX-WS" installation directory.

You just need to install "ant" http://ant.apache.org/manual/install.html
 and set "AS_HOME " to point to the Application Server installation
directory.

Then run the Application server , next build the Server and Client
side components.

Samples for Message Handler

NetBean Sample (you don't need to worry about Handler.xml):

http://qa.netbeans.org/modules/j2ee/promo-g/end2end/hello_ws.html#creatingHandler

Sample codes coming together with JAX-WS RI:

 fromjavahandler, efficient_handler,....

Other samples in which you may need to override service deployment description webservices.xml:

http://cit3.ldl.swin.edu.au/wikka/wikka.php?wakka=WebServiceshttp://blog.jdevelop.eu/2008/03/08/how-to-modify-jax-ws-soap-messages/

Homework #2 – Part 3

Tasks and Deliverables

Notice!

You can not move to Homework 2 unless you are done with Homework1 !!!

Tasks (1)

Design , implement , deploy and test web services for University, Employment Office, Company Database , Employment Service Company and Recruiting Company such that:

1- University, Employment Office and Company Database provides a service to get the requested documents (Transcript, Employment Record and Company Info) as XML document.

2. Job Service Company provides a service accepting a short CV of an applicant, and then it utilizes the above mentioned services to get other necessary documents. The services provides (returns) the generated Profile as an XML document.

Tasks (2)

3- To simplify the process, assume that Employment Service Company has already interviewed the applicant, and it has the additional information (e.g. References, Cities desire to work,...)

4- Develop a web service for a recruting companies such that the service returns their advertised job descriptions which matches the given criteria (e.g key words). The criteria is service input. You need to define a simple XSD for advertised job description.

5- You SHOULD develop a test client for each Web Service inorder to show that each service wrorks indpendently.

6- Combine all the the above mentioned services into a system such that by giving the Job Service Company the short CV of applicant, it returns the complete applicant profile.

Service Interactions
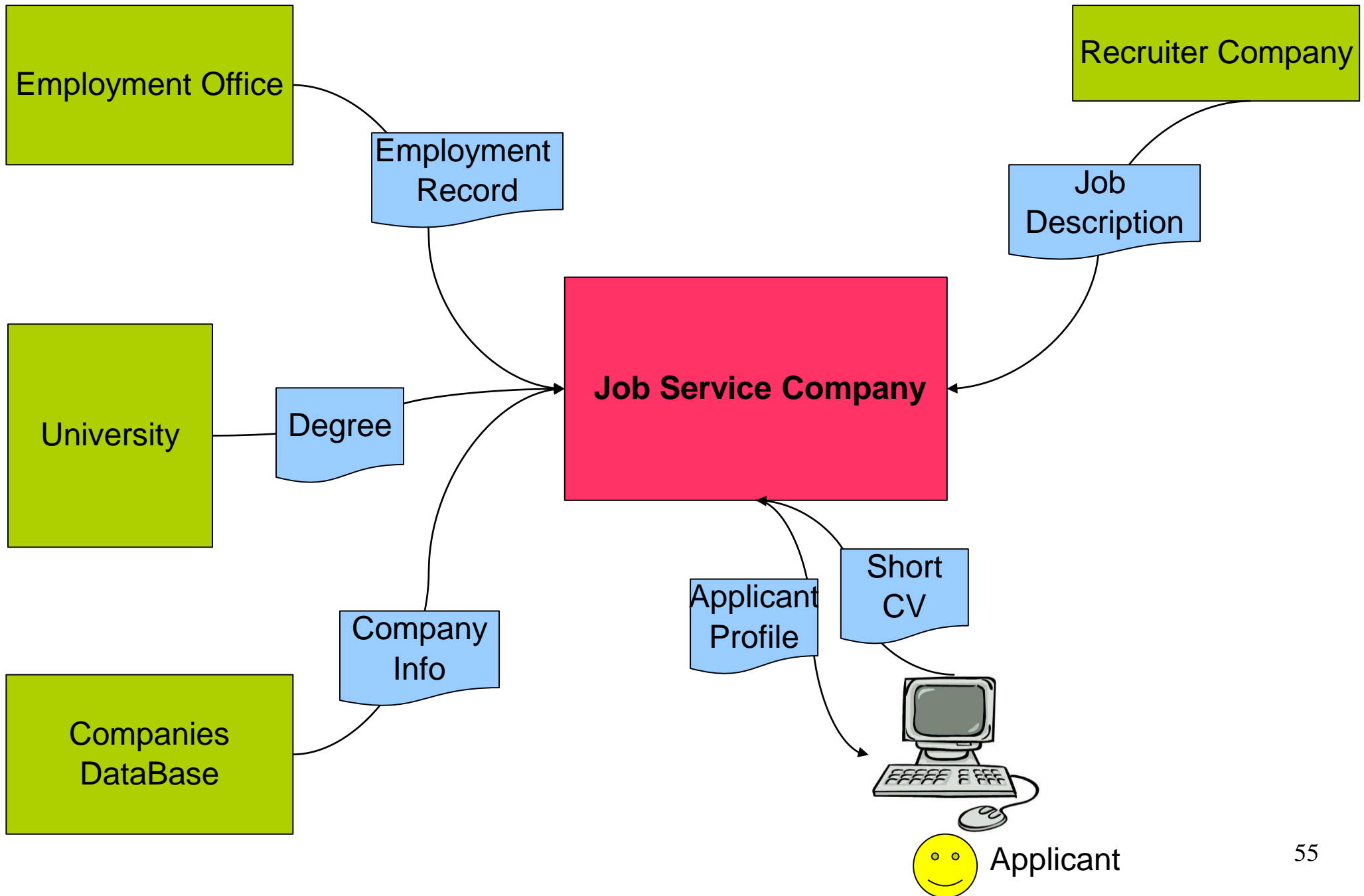
Points that you should consider (1)

1- You can use any of the schema mapping approaches (DOM, SAX,....) you have already developed in HW#1.

2. The style of designed web services should be mixture of both "Document" styles .

Points that you should consider (2)

5. Design only One of the services in both Synchronous and Asynchronous mode. Explain your approach for Asynchronous service very short but precise (max 3 lines)

6. It is not compulsory to use "SOAP Message Handler" approach. You can choose another way.

Deliverables

1- The Source code, WSDLs and Schema of the implemented Web services.

2- The XML of constructed, sent and received SOAP messages communicated among services.

3- A short and precise description showing your architecture and deployment of your web services.

4. Executable version of your system

5. Show your fully functional system in a 10-15 minutes presentation.

Good Luck!

HW #2 - Delivery

Send your deliverables by e-mail to BOTH:

shahabm@kth.se and nimad@kth.se

e-mail subject: PWS10-HW2

Please add your names in the body of the email

Attach: source code + instructions how to run your code,

Deadline : 21-02-2010, 11:59 PM CET

Presentation: 19 Feb 2009 (somewhere in 6th floor)

References and Tutorials

– IBM JAX-WS tutorial

https://www6.software.ibm.com/developerworks/education/ws-jax/ws-jax-a4.pdf

– SUN WSDP tutorial for WSDL, SOAP and SAAJ programming.

http://java.sun.com/webservices/docs/2.0/tutorial/doc/

http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/

– SOAPUI tools to monitor SOAP messages:

http://www.soapui.org/

– Oracle SOAP Message Handler:

http://java.sun.com/mailers/techtips/enterprise/2006/TechTips_Aug06.html#2

http://edocs.bea.com/wls/docs103/webserv_adv/handlers.html#wp222336

Useful Links

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/twbs_jaxwsdeploydescriptor.html

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/twbs_deployapp2.html

Links for message handlers:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/twbs_jaxwshandler.html

NetBeans Tutorials for developing web services and adding message handler.

http://www.javapassion.com/planning/handsonbyol/netbeanswebservices/

http://qa.netbeans.org/modules/j2ee/promo-g/end2end/hello_ws.html

http://www.netbeans.org/kb/docs/websvc/jax-ws.html

http://qa.netbeans.org/modules/j2ee/promo-g/end2end/hello_ws.html#creatingHandler

http://www.netbeans.org/kb/60/websvc/wsdl-guide.html

How to invoke a service

```
try {
          Service service = new Service();
          Call call = (Call) service.createCall();

call.setTargetEndpointAddress("http://localhost:18181/SynchronousSampleSer
vice/SynchronousSamplePort");

          call.setOperationStyle(Style.RPC);
          call.setOperationName(new QName(NAMESPACE,
"SynchronousSampleOperation"));

          call.addParameter("a", Constants.XSD_INT, ParameterMode.IN);
          call.addParameter("b", Constants.XSD_INT, ParameterMode.IN);
          call.addParameter("r", Constants.XSD_INT, ParameterMode.OUT);
          call.setReturnType(Constants.XSD_ANY);
          String result = call.invoke(new Object[]{"1000",
"123456"}).toString();
          System.out.println(result);
      } catch (Exception ex) {
          ex.printStackTrace();
      }
```

62

Question?