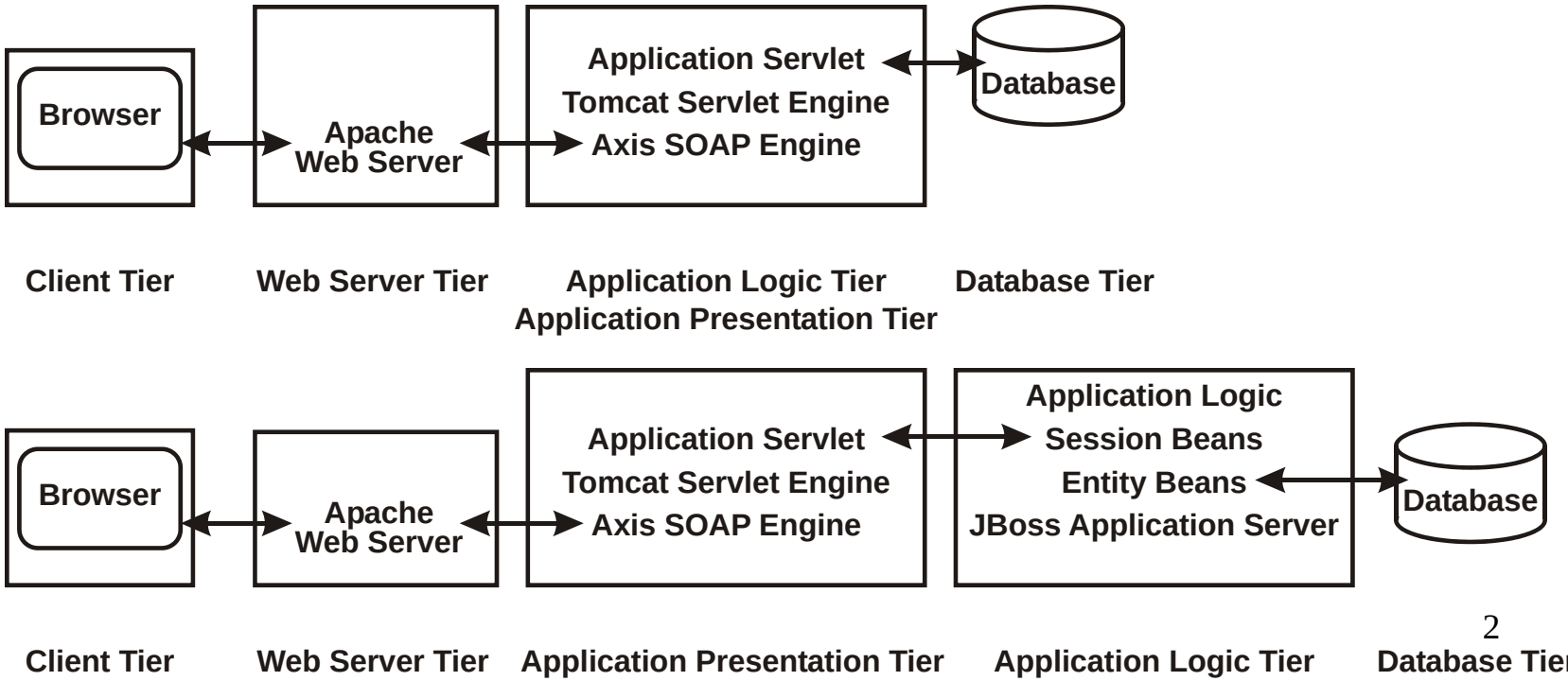
Network Computing Lecture 17

Professor Louise E. Moser

Spring 2012

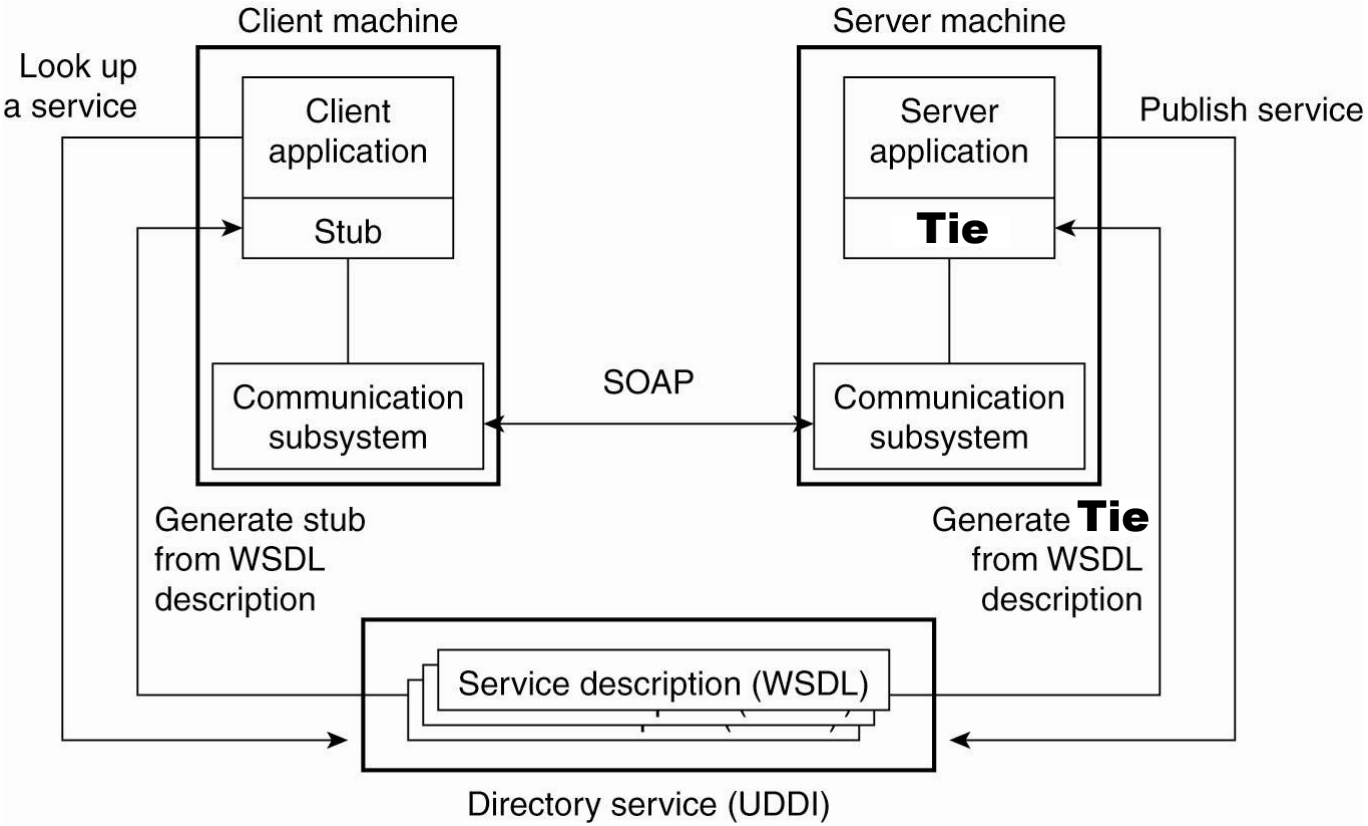
Web Services

- A Web Service is a server-side application that implements procedures that clients can call
- A Web Service is deployed in a Servlet container using Apache Tomcat, or in an EJB container using a J2EE application server such as JBoss



Web Services

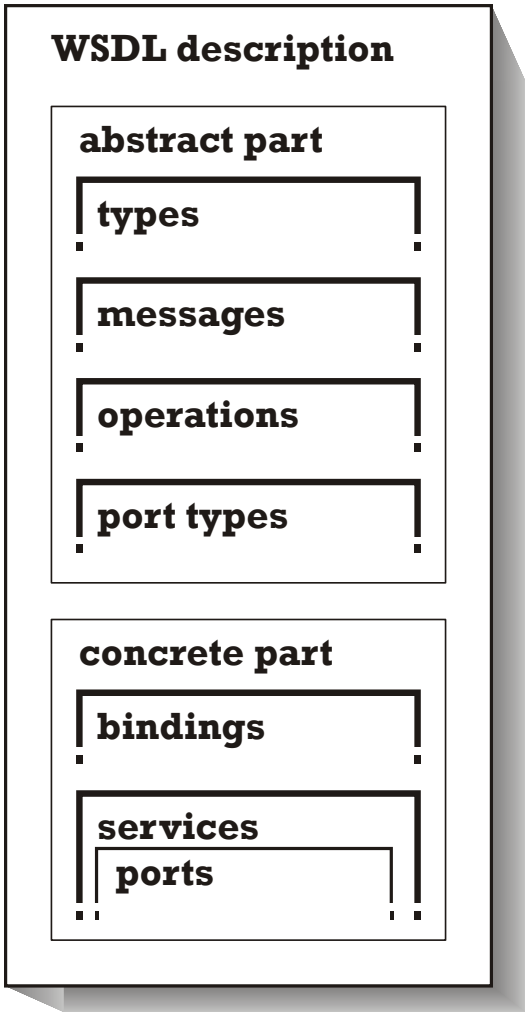
- The organization of a Web Service, based on the XML, SOAP, WSDL, and UDDI standards



Web Services Description Language

- A Web Service can make itself available to clients via a **Web Services Description Language (WSDL)** document that provides:
  - Name of the Web Service
  - Operations that can be called on it
  - Parameters of those operations
  - Location to send requests
- A Web client can use a WSDL document to discover what the service offers and how to access it
- A Web client can go to a business registry to get a WSDL document for a particular service that it needs

Web Services Description Language Document



Simple Object Access Protocol

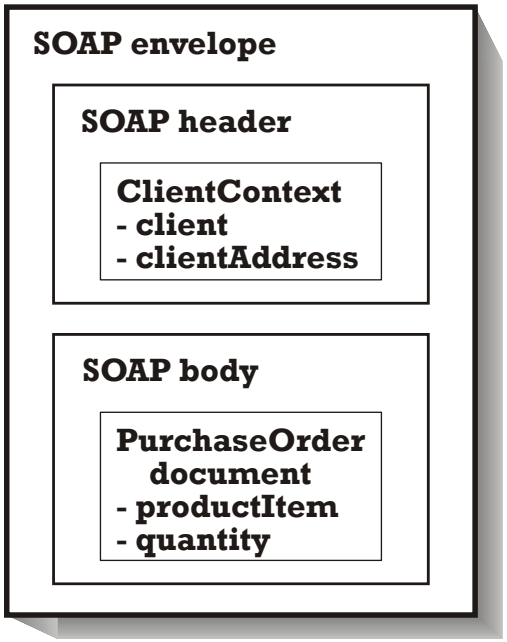
- The **Simple Object Access Protocol (SOAP)** is an application-layer protocol for invoking procedures or exchanging XML documents
- SOAP typically operates over HTTP, but it can also operate over the Simple Mail Transfer Protocol (SMTP)
- The SOAP specification, available at *http://www.w3.org/* , defines
  - What is required and optional in a SOAP message
  - How data can be encoded and transmitted in a SOAP message
- Both **JAX Remote Procedure Call (JAX-RPC)** and **SOAP with Attachments API for Java (SAAJ)** use SOAP
  - Both JAX-RPC and SAAJ are based on the request / response synchronous (blocking) model
    - However, both JAX-RPC and SAAJ also support asynchronous communication and one-way messages
  - SAAJ is the lower-level equivalent of JAX-RPC

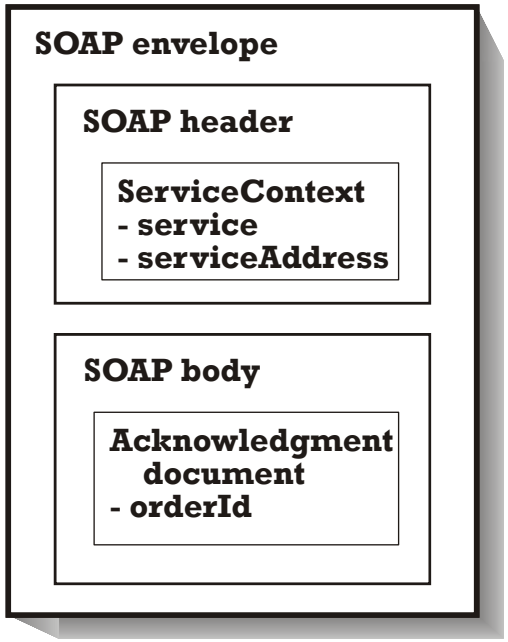
SOAP Message Object

- A ***SOAPMessage*** object has a required SOAP Part, and may also have one or more Attachment Parts
- The ***SOAP Part*** must have a ***SOAPEnvelope*** object, which may contain a *SOAPHeader* object and must contain a *SOAPBody* object
  - The ***SOAPHeader*** object can include one or more headers
  - The ***SOAPBody*** object can hold XML fragments as the content (payload) of the message being sent
- An ***Attachment Part*** can be used to send content that is not in XML format, or that is a fragment of an XML document
  - The attachment part can include any kind of content, such as audio, video and image data, as well as XML-formatted data

Structure and Content of a SOAP Message

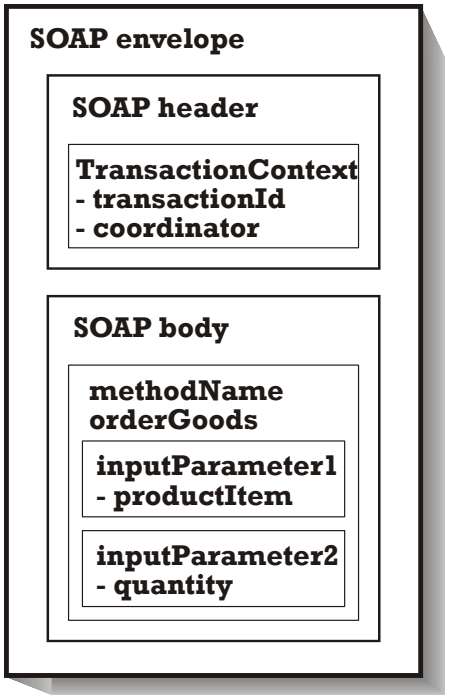
- **Document-oriented**
  - At the left, a SOAP **request** message with an **XML document** in the SOAP body
  - At the right, a SOAP **response** message with an **XML document** in the SOAP body

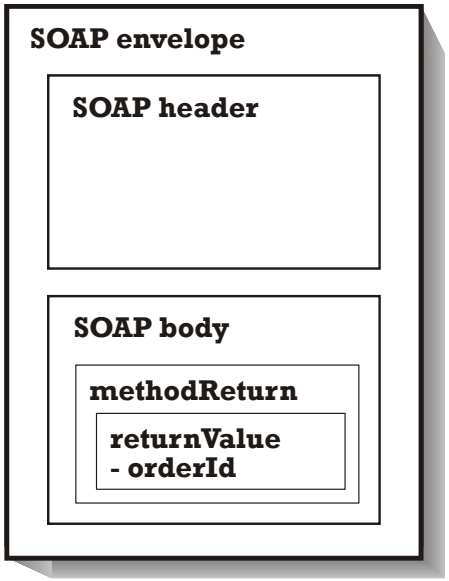




Structure and Content of a SOAP Message

- **Procedure-oriented**
  - At the left, a SOAP **request** message with a **method invocation** in the SOAP body
  - At the right, a SOAP **response** message with a **return value** in the SOAP body





Steps in Using SOAP

- Getting a Connection
- Creating a Message
- Populating the SOAP Part of a Message
- Populating the Attachment Part of a Message
- Sending a Message

Getting a Connection

- First, a client must get a connection in the form of a ***SOAPConnection*** object, which is a point-to-point connection from the sender to the receiver
- The SOAP Connection is created by a ***SOAPConnectionFactory*** object
- A client obtains the default *factory* object as follows:

```
SOAPConnectionFactory factory =
                    SOAPConnectionFactory.newInstance();
```

- The client then uses the *factory* object to create a *connection* object

```
SOAPConnection connection =
                    factory.createConnection();
```

Creating a Message

- First, a sender must get a ***MessageFactory*** object
- The sender obtains the default *messageFactory* object as follows:

  MessageFactory messageFactory =
  MessageFactory.newInstance();

- The sender then uses the *messageFactory* object to create a ***SOAPMessage*** object

  SOAPMessage ***message*** =
  messageFactory.createMessage();

- The *message* object automatically contains the required SOAP Part, i.e., *SOAPEnvelope* and *SOAPBody*, as well as the optional *SOAPHeader* (which is included for convenience)
- The *SOAPHeader* and *SOAPBody* objects are initially empty, and you need to populate them

Populating the SOAP Part of a Message

- One way to add content to the SOAP Part of a message is to create a ***SOAPHeaderElement*** object or a ***SOAPBodyElement*** object and add an XML fragment using the ***addTextNode()*** method, as in:

```
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getSOAPEnvelope();
SOAPBody body = envelope.getSOAPBody();
SOAPBodyElement bodyElement = body.addBodyElement(
                              envelope.createName("text", "hotitems",
                              "http://hotitems.com/products/gizmo");
bodyElement.addTextNode("some-xml-text");
```

Populating the SOAP Part of a Message (cont)

- Another way to add content to the SOAP Part of a message is to pass it a ***javax.xml.transform.Source*** object, which contains content for the SOAP Part and the information needed for it to act as source input
- Such an object may be a *SAXSource*, *DOMSource*, or *StreamSource* object
  - The ***SAXSource*** and ***DOMSource*** objects contain content and instructions for transforming the content into an XML document
  - The ***StreamSource*** object contains the content as an XML document

Populating the SOAP Part of a Message (cont)

```
SOAPPart soapPart = message.getSOAPPart();
DocumentBuilderFactory dbFactory =
                                DocumentBuilderFactory.newInstance();
DocumentBuilder builder = dbFactory.newDocumentBuilder();
Document document = builder.parse("file://foo.bar/soap.xml");
DOMSource domSource = new DOMSource(document);
soapPart.setContent(domSource);
```

- To set the content of the SOAP message, you use the ***soapPart.setContent()*** method, as above

Populating the SOAP Part of a Message (cont)

- If you want to send an XML document as the content of a SOAP message, you can use the ***soapBody.addDocument()*** method, as follows

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

- This feature allows you to keep your application data in a document that is separate from the SOAP Envelope, until it is time to send the message

Populating the Attachment Part of a Message

- A *SOAPMessage* object has 0 or more Attachment Parts, which may contain anything from plain text to images
- In the following example, the content is an image in a JPEG file, whose URL is used to initialize the ***javax.activation.DataHandler*** object handler

```
URL url = new URL("http://foo.bar/img.jpg");
DataHandler handler = new DataHandler(url);
Attachmentpart attachPart = message.createAttachmentPart(handler);
message.addAttachmentPart(attachPart);
```

Populating the Attachment Part of a Message (cont)

- Content can also be given to an Attachment Part by passing an object and its content type to the ***message.createAttachmentPart()*** method, as in:

```
AttachmentPart attachPart =
    message.createAttachmentPart("content-string", "text/plain");
message.addAttachmentPart(attachPart);
```

Sending a Message

- A client uses the ***soapConnection.call()*** method to send a message
- The *soapConnection.call()* method **sends** the message and then **blocks** until it gets a **response**, using the **synchronous** request/response kind of interaction
- The arguments of the *soapConnection.call()* method are the message being sent and a URL object that contains the URL of the receiver endpoint

SOAPMessage ***response*** = ***soapConnection.call(message, endpoint);***

SOAP Faults

- If a message is sent unsuccessfully (e.g., because of an incomplete address), you might get a response containing a ***SOAPFault*** element with status or error information
  - There can be at most one *SOAPFault* element in a message, and it is included in the *SOAPBody*
  - If there is a *SOAPFault* element in the *SOAPBody*, there can be no other elements in the *SOAPBody*
- A *SOAPFault* object is similar to an *Exception*, but
  - It is an element in a message's *SOAPBody*, rather than part of the *try / catch* mechanism used for an *Exception*
  - It reports only the status or error information and, unlike an *Exception*, does not halt execution of the application

SOAP

- An example of an XML-based SOAP message

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
      <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
        <n:priority>1</n:priority>
        <n:expires>2001-06-22T14:00:00-05:00</n:expires>
      </n:alertcontrol>
    </env:Header>
    <env:Body>
      <m:alert xmlns:m="http://example.org/alert">
        <m:msg>Pick up Mary at school at 2pm</m:msg>
      </m:alert>
    </env:Body>
</env:Envelope>
```

Web Services

- **Web Service** – Software designed to support interoperable machine-to-machine interactions over a network
- **Two major classes of Web Services**
  - **Simple Object Access Protocol (SOAP)** – Service is exposed via an arbitrary set of operations, which are conveyed in XML-formatted SOAP messages over HTTP
  - **REpresentational State Transfer (REST)** – Service uses a uniform set of stateless operations (HTTP methods) to manipulate XML representations of Web resources, which are conveyed directly over HTTP

REST Web Services

- **REpresentational State Transfer (REST) Web Services**
  - Based on the notion of a Resource with a Uniform Resource Identifier (URI)
    - Services and servers are resources; clients may or may not be resources
  - Are lighter weight than SOAP Web Services
  - Require little infrastructure support other than HTTP and XML
  - Use the standard HTTP methods
    - GET – Obtain a representation of a resource
    - DELETE – Remove a representation of a resource
    - POST – Update or create a representation of a resource
    - PUT – Create a representation of a resource
  - Messages are in XML format, but are confined to a particular schema written in a schema language like XML Schema
  - Messages can be encoded using URL encoding

REST Web Services

- **REpresentational State Transfer (REST) Web Services**
  - Descriptors are written in the **Web Application Description Language (WADL)**
  - Web Application Description Language (WADL) is similar to the Web Services Description Language (WSDL)
  - A WADL descriptor describes the service, including the attributes and methods of the service
  - WADL also aids in the construction of stubs that are used with the service clients