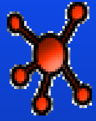
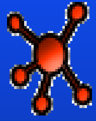




# SOAP

## High-level Data Exchange!

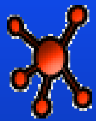
Part of the code shown in the following slides is taken from the book "Java Web Services" by D.A. Chappell and T. Jawell, O'Reilly, ISBN 0-596-00269-6





# SOAP History

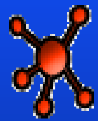
- SOAP 1.0 (1997):
  An XML-based protocol for accessing objects
- XML-RPC (1998): A subset of SOAP 1.0
- SOAP 1.1 (2000): Widely supported, de facto standard
- SOAP 1.2 (2003, june): W3C "Recommendation"
  - Standard, will soon replace SOAP 1.1 implementations
- Original purpose: interoperable protocol for accessing "objects"
- Current versions:
  focus on a generalized XML messaging framework





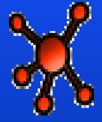
# SOAP Features

- Extensibility
  - Allows addition of features as layered extensions
- WS-Security, WS-Routing, …
- Usable over a variety of transport protocols
  - TCP, HTTP, SMTP, etc.
  - Standard protocol bindings need to be defined:
    i.e., specs on how a SOAP msg is encoded in each protocol
- Support for a variety of programming models
  - RPC-like request-response
  - One-way messaging
  - etc.



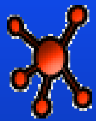
# Elements of the SOAP Specification

- Messaging framework
  - ***a suite of XML elements for "packaging" XML messages*** to be exchanged between systems
  - i.e., what a "SOAP message" is actually made of
- Protocol bindings
  - Rules for ***transmission of SOAP msg*** upon a given transport means
  - typical HTTP binding
- RPC encoding
  - Standard way for ***mapping RPC calls onto SOAP messages***
- Processing model
  - Permits ***multiple intermediary nodes*** to act upon msg
  - Rules for handling a SOAP msg along the path from sender to receiver





# Messaging Framework

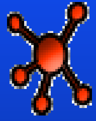


Web Services

# SOAP Messaging Framework

- Core XML elements: Envelope, Header, Body, and Fault
  - Defined in a version-specific XML namespace
    - SOAP 1.1: http://schemas.xmlsoap.org/soap/envelope
    - SOAP 1.2: http://www.w3.org/2003/05/soap-envelope
- Structure of a SOAP message

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV ="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header> <!-- optional -->
    <!-- header blocks go here … -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <!-- payload or Fault element goes here …-->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



*Web Services*

# Structure of a SOAP Message

**SOAP message (an XML document)**

**SOAPPart**

**SOAPEnvelope**

**SOAPHeader (optional)**

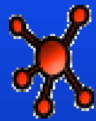
**header**

**header**

**SOAPBody**

**XML content**

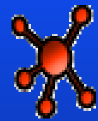
**SOAPFault (opt.)**



Web Services

# To Make XML Become a SOAP Message

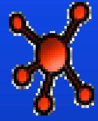
1. Wrapping of the XML document inside a SOAP body;
2. Wrapping of a SOAP body within a SOAP envelope;
3. Optional inclusion of a SOAP header block;
4. Namespace declarations;
5. Encoding style directives for the data marshalling
6. Binding of all this stuff to a protocol





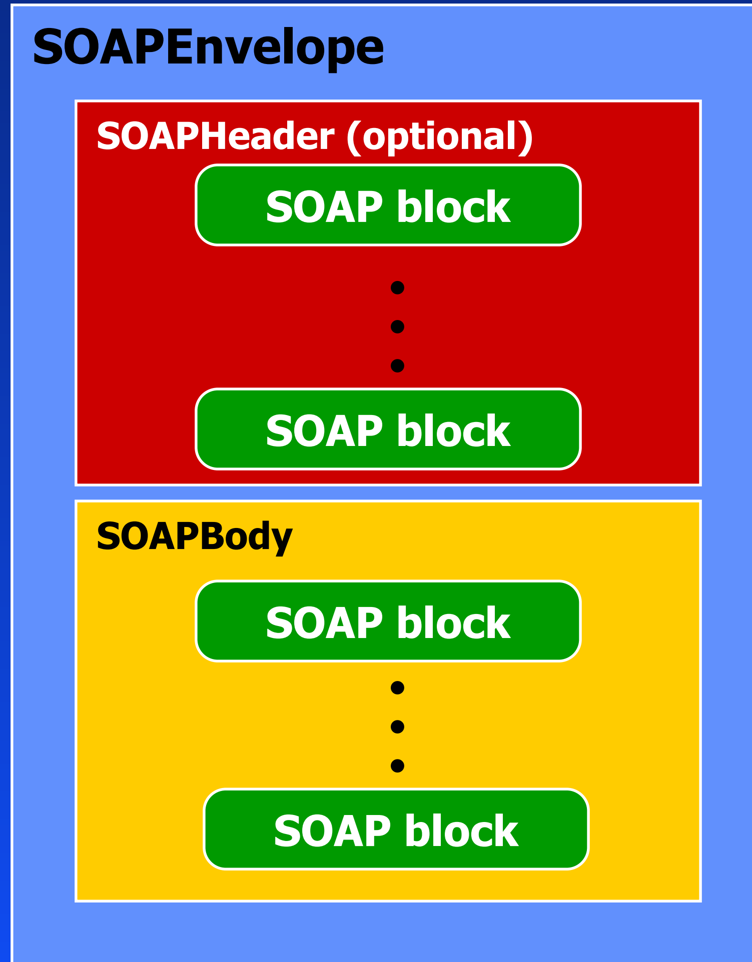
# An XML Document…

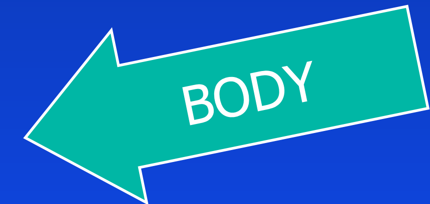
```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder xmlns="urn:oreilly-jaws-samples">
   <shipTo country="US">
        <name>Joe Smith</name>
        <street>14 Oak Park</street>
        <city>Bedford</city>
        <state>MA</state>
        <zip>01730</zip>
   </shipTo>
   <items>
        <item partNum="872-AA">
               <productName>Candy Canes</productName>
               <quantity>444</quantity>
               <price>1.68</price>
               <comment>I want candy!</comment>
        </item>
   </items>
</PurchaseOrder>
```

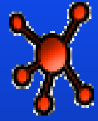


Web Services

# Block Structure of a SOAP Envelope

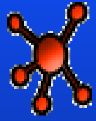






Web Services

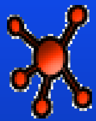
# Messaging Framework: Transmitting XML Docs



Web Services

# ...Wrapping the XML Document into the Envelope...

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Header>
...
</SOAP-ENV:Header>

<SOAP-ENV:Body>
   <PurchaseOrder xmlns="urn:oreilly-jaws-samples">
   ...
   </PurchaseOrder>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

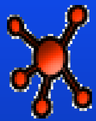


Web Services

# …Wrapping the XML Document…

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Header>
...
</SOAP-ENV:Header>

<SOAP-ENV:Body>
   <PurchaseOrder xmlns="urn:oreilly-jaws-samples">
   ...
   </PurchaseOrder>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Namespace declarations

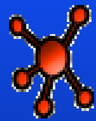


# The SOAP Header

- SOAP 1.1 and SOAP 1.2 have no conventions on the header's contents – *the SOAP Header is optional!*
- The header is just the place to put directives to the SOAP processor that receives the message
- The send and receiving parties need to agree on the header's contents (and their semantics)

Example:

```
<SOAP-ENV:Header>
    <jaws:MessageHeader
        xmlns:jaws="urn:oreilly-jaws-examples">
            <MessageID>1453</MessageID>
    </jaws:MessageHeader>
<SOAP-ENV:Header>
```

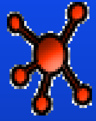




# SOAP Header Contents

- The SOAP Header can contain one or more sub-elements (called *header blocks*)
  - Each header block can be an element from some namespace
  - Attribute **mustUnderstand**="1" indicates receiver **must** understand this header block (mandatory)
  - Else return a Fault element
- **Primary source for extensibility**
  - Security tokens, routing info, processing instructions, ...

```
<SOAP-ENV:Header>
     <!-- security credentials -->
     <s:credentials  xmlns:s="urn:examplesorg:security"
                     soap:mustUnderstand="1" >
            <username>alex</username>
            <password>goofy</password>
     </s:credentials>
</SOAP-ENV:Header>
```

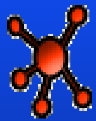


# HTTP Protocol Binding

Purpose of the HTTP protocol binding: **two-fold**

- To ensure that SOAP is carried in a way that is consistent with HTTP's message model
  - Intent is not to break HTTP
- To indicate to HTTP servers that this is a SOAP message
  - Allows HTTP servers to act on a SOAP message without knowing SOAP

- Binding *usually* works for HTTP POST requests



Web Services

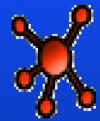
# Relying on the HTTP Request

- Use HTTP POST request method name
- Use the SOAPAction HTTP header field
  - It cannot be computed – the sender must know
  - *It should indicate the intent – not the destination*
- SOAP request doesn't require SOAP response

SOAP Processor / Router

```
POST /Accounts/Henry HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction:
"http://electrocommerce.org/MyMessage"

<SOAP:Envelope...
```

# The SOAP Protocol Binding for Our Message

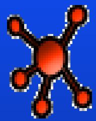


Web Services

```
SOAPAction = "urn:soaphttpclient-action-uri"
Host = localhost
Content-Type = text/xml; charset=utf-8
Content-Length = 701
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
...
</SOAP-ENV:Envelope>
```

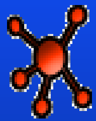
- The intent of the SOAPAction header is to provide the HTTP receiver *info to route/dispatch the message*, with no need to parse the SOAP envelope
- SOAPAction is required in SOAP 1.1 and optional in SOAP 1.2





# A Sender for Our Message (I)

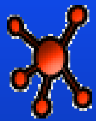
```
public void sendSOAPMessage(){
  try {
    // get soap body to include in the SOAP envelope
    FileReader fr = new FileReader (m_dataFileName);

    javax.xml.parsers.DocumentBuilder xdb =
       org.apache.soap.util.xml.XMLParserUtils.getXMLDocBuilder();

    org.w3c.dom.Document doc =
       xdb.parse (new org.xml.sax.InputSource (fr));

    if (doc == null) {
       throw new org.apache.soap.SOAPException
         (org.apache.soap.Constants.FAULT_CODE_CLIENT, "parsing error");
    }
    // create a vector for collecting the header elements
    Vector headerElements = new Vector();

    // Create a header element in a namespace
    org.w3c.dom.Element headerElement =
       doc.createElementNS(URI,"jaws:MessageHeader");
    //filling of header part: omitted
    headerElements.add(headerElement);
```



Web Services

# A Sender for Our Message (II)

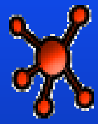
```
//Create the SOAP envelope
org.apache.soap.Envelope envelope = new org.apache.soap.Envelope();

// create a vector for collecting the body elements
Vector bodyElements = new Vector();

//obtain the top-level DOM element and place it into the vector
bodyElements.add(doc.getDocumentElement ());

//Add the SOAP header element to the envelope
org.apache.soap.Header header = new org.apache.soap.Header();
header.setHeaderEntries(headerElements);
envelope.setHeader(header);

//Create the SOAP body element
org.apache.soap.Body body = new org.apache.soap.Body();
body.setBodyEntries(bodyElements);

//Add the SOAP body element to the envelope
envelope.setBody(body);
```





# A Sender for Our Message (III)

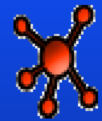
```
    // Build the Message.
    org.apache.soap.messaging.Message msg =
       new org.apache.soap.messaging.Message();

    msg.send (new java.net.URL(m_hostURL), URI, envelope);

    // receive response from the transport and dump it to the screen
    org.apache.soap.transport.SOAPTransport st =msg.getSOAPTransport ();

    BufferedReader br = st.receive ();
    String line = br.readLine();
    if(line == null) {
       System.out.println("HTTP POST was successful. \n"); }
    else {
       while (line != null) { System.out.println (line);
          line = br.readLine();
       }
    }
  }
  catch(Exception e) { e.printStackTrace(); }
}
```



Web Services

# A Receiver (Servlet) for Our Message (I)

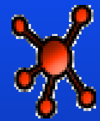
```
// Our SOAP requests are going to be received as HTTP POSTS
public void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws IOException, ServletException {

    // Traverse the HTTP headers and show them on the screen
    for( Enumeration enum = request.getHeaderNames();
         enum.hasMoreElements(); ) {
       String header = (String)enum.nextElement();
       String value  = request.getHeader(header);
       System.out.println("   " + header + " = " + value);
    }
```



# A Receiver (Servlet) for Our Message (II)

```
  // If there's anything in the msg body, dump it to the screen
  if(request.getContentLength() > 0) {
     try {
        java.io.BufferedReader reader = request.getReader();
        String line = null;
        while((line = reader.readLine()) != null) {
           System.out.println(line);
        }
     } catch(Exception e) { System.out.println(e); }
  }

  response.setContentType("text/xml");
}
```

- And what about SOAP here???
- This servlet is not “SOAP-aware”!!!
  (but it can “plainly” process SOAP payloads...)



Web Services

# Exploiting the (Apache) SOAP Message Routing Service

SOAP msg Sender

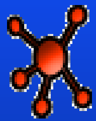
to: URL msgRouter
SOAPAction=…

SOAP-enabled Web Server (Apache, Tomcat)

SOAP Message Router (servlet)

Msg Acceptor

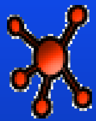
Deployment Descriptor for **Msg Acceptor**



Web Services

# Deployment Descriptor: Contents

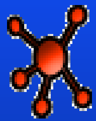
```
<isd:service xmlns:isd=“”
  id=“urn:oreilly-jaws-samples” type=“message”>
 <isd:provider
     type=“java”
     scope=“Application”
     methods=“PurchaseOrder”>
   <isd:java class=“PurchaseOrderAcceptor”/>
 </isd:provider>
 <isd:faultListener>…</isd:faultListener>
</isd:service>
```



Web Services

# A "Message Acceptor" (I)

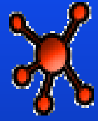
```
public void PurchaseOrder(Envelope requestEnvelope,
                          SOAPContext requestContext,
                          SOAPContext responseContext)
                          throws SOAPException    {
   System.out.println("Received a PurchaseOrder!!");
   ...
   org.apache.soap.Header header = requestEnvelope.getHeader();
   Vector headerEntries = header.getHeaderEntries();

   for (Enumeration e = headerEntries.elements();
  e.hasMoreElements();)   {
       org.w3c.dom.Element el = (org.w3c.dom.Element)e.nextElement();
       ...
       // process mustUnderstand
       String mustUnd=el.getAttribute("SOAP-ENV:mustUnderstand");
       if (mustUnd!=null) {...} else {...}
       String tagName = el.getTagName();
       if(tagName.equalsIgnoreCase("jaws:MessageHeader")) {...}
   }
```



Web Services

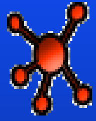
# A "Message Acceptor" (II)

```
org.apache.soap.Body body = requestEnvelope.getBody();
  java.util.Vector bodyEntries = body.getBodyEntries();

for (Enumeration e = bodyEntries.elements(); e.hasMoreElements();) {

 org.w3c.dom.Element el = (org.w3c.dom.Element)e.nextElement();

 org.apache.soap.util.xml.DOM2Writer.serializeAsXML((org.w3c.dom.Node)
 el, writer);
}
System.out.println(writer.toString());

try {
  responseContext.setRootPart(
        "<PurchaseOrderResponse>Accepted</PurchaseOrderResponse>",
        "text/xml");
}
catch(Exception e) {...}
}
```





# Messaging Framework: SOAP with Attachments



Web Services

# SOAP with Attachments

SOAP message

SOAPPart

SOAPEnvelope

SOAPHeader (optional)

header

SOAPBody

XML content

SOAPFault (opt.)

AttachmentPart (optional)

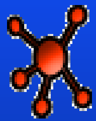
MIME header

Content (XML or not)

AttachmentPart (optional)

MIME header

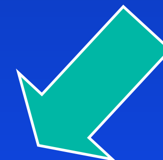
Content (XML or not)

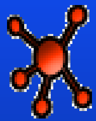




# Document to send

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrderWithAttachment xmlns="urn:oreilly-jaws-
   samples">
   <shipTo country="US">
         ...
   </shipTo>
   <items>
         ...
   </items>
   <attachment href="cid:the-attachment"/>
</PurchaseOrderWithAttachment>
```

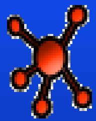




Web Services

# Building up SwA messages

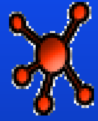
```
...
// Build the Message.
org.apache.soap.messaging.Message msg =
                  new org.apache.soap.messaging.Message();

//Attach any attachments
   if(m_att != null) {
     BufferedReader attReader =
               new BufferedReader(new FileReader(m_att));
     StringBuffer buffer = new StringBuffer();
     for(String line = attReader.readLine(); line != null;
             line = attReader.readLine()) {
       buffer.append(line);
     }
     MimeBodyPart attachment = new MimeBodyPart();
     attachment.setText(buffer.toString());
     attachment.setHeader("Content-ID", "the-attachment");
     msg.addBodyPart(attachment);
   }
msg.send (new java.net.URL(m_hostURL), URI, envelope);
...
```



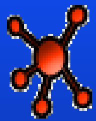


# Receiving SwA messages

```
...
for (Enumeration e = bodyEntries.elements(); e.hasMoreElements();) {

     org.w3c.dom.Element el = (org.w3c.dom.Element)e.nextElement();
     org.apache.soap.util.xml.DOM2Writer.serializeAsXML(
                                  (org.w3c.dom.Node)el, writer);
     org.w3c.dom.Element attachmentEl =
       (org.w3c.dom.Element)el.getElementsByTagName("attachment").item(0);

     if (attachmentEl != null) {

        cid = attachmentEl.getAttribute("href").substring(4);
        writer.write("Content-ID = "+cid+"\n");
        MimeBodyPart attachment = requestContext.getBodyPart(cid);
        try {
           writer.write("The attachment is...\n" +
  attachment.getContent()+"\n");
        }
        catch(Exception ex) { ... }

     }else
         writer.write("The Content-ID is null!\n");
}
...
```



# Messaging Framework: SOAP RPC



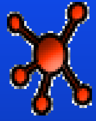
# RPC Request

- “Encoding” defines how to encapsulate RPC info *within the SOAP body*
  - Endpoint location (URI), procedure(method) name, parameter names & values
- A procedure call is modeled as an XML struct, whose root element corresponds to the procedure name
  - Named fields for each in or in/out parameter

Procedure name

```
<SOAP-ENV:Body>
<myns:GetPrice xmlns:myns=“urn:xmethods-BNPriceCheck“
   SOAP-ENV:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/">
      <isbn xsi:type=“xsd:string”>0596000686</isbn>
</myns:GetPrice >
</SOAP-ENV:Body>
```

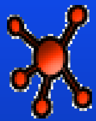
Parameter





# RPC Encoding Attribute

- The SOAP encoding style consists of a set of rules to be applied for the coding (serialization/marshalling) of transmitted data
- The *encodingStyle* attribute refers to an XML schema that describes with the particular rules to be applied
- XML schema for SOAP 1.1: http://schemas.xmlsoap.org/soap/encoding
- XML schema for SOAP 1.2: http://www.w3.org/2001/09/soap-encoding



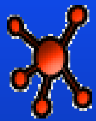


# RPC Response

- The Method response also modeled as a struct, named *MethodName*Response

```
<soap:Body
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <n:getLocationRequest>
      <ip xsi:type="xs:string">131.114.9.4</ip>
   </n:getLocationRequest>
</soap:Body>
```

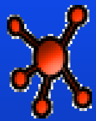
```
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
   <ns1:getLocationRequestResponse>
      <return xsi:type="xsd:string">EU</return>
   </ns1:getLocationRequestResponse>
</SOAP-ENV:Body>
```



Web Services

# RPC: client (I)

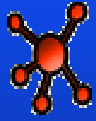
```
public void checkStock() throws Exception {

    org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call();

    //This service uses standard SOAP encoding
    String encStyleURI = org.apache.soap.Constants.NS_URI_SOAP_ENC;
    call.setEncodingStyleURI(encStyleURI);

    call.setTargetObjectURI ("urn:stock-onhand"); //Set the target URI

    call.setMethodName("getQty"); //Set the method name to invoke

    //Create the parameter objects
    Vector params = new Vector ();
    params.addElement(
         new org.apache.soap.rpc.Parameter(
                                    "item", String.class, m_item, null)
        );
    //Set the parameters
    call.setParams(params);
```





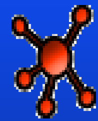
# RPC: client (II)

```
//Invoke the service
org.apache.soap.rpc.Response resp
      = call.invoke ( new java.net.URL(m_hostURL),"");

//Check the response
if (resp != null) {
  if (resp.generatedFault ()) {
    org.apache.soap.Fault fault = resp.getFault ();
    System.out.println ("Call failed due to a SOAP Fault: ");
    System.out.println ("Fault Code=" + fault.getFaultCode ());
    System.out.println ("Fault String=" + fault.getFaultString ());
  }
  else {
    org.apache.soap.rpc.Parameter result = resp.getReturnValue ();
    Integer intresult = (Integer) result.getValue();
    System.out.println ("The quantity is: " + intresult );
   }
 }
}
```



# RPC: server

```
...

public class StockQuantity{

  public int getQty (String item)
    throws org.apache.soap.SOAPException {

    int inStockQty = (int)(Math.random()*(double)1000);

    return inStockQty;
   }
   ...
}
```



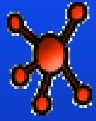
# RPC server: Deployment descriptor

- The WS class is handled by the WS container; all the required info must be available to the container through its deployment descriptor

```
<isd:service
    xmlns:isd=http://xml.apache.org/xml-soap/deployment
    id="urn:stock-onhand">
  <isd:provider type="java"
                scope="Application"
                methods="getQty">
      <isd:java class="StockQuantity"/>
  </isd:provider>
  <isd:faultListener>
     org.apache.soap.server.DOMFaultListener
   </isd:faultListener>
</isd:service>
```

Method name

implementation

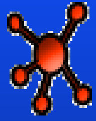


Web Services

# Parameter Encoding: Data Types

- Parameter encoding *relies on XML schemas* (namespaces xsd: xsi: )

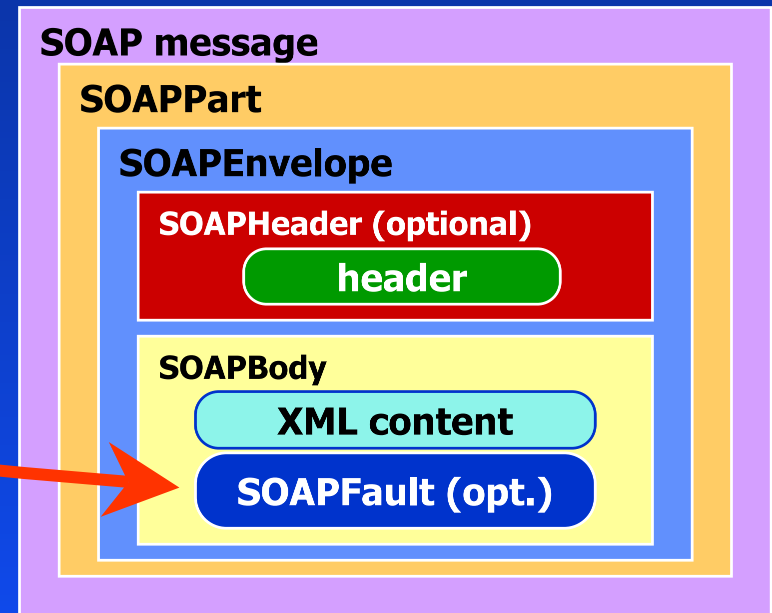
- Simple built-in types
- Arrays
- Enumeration of simple built-in types
- XML structs & Complex types

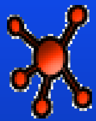


*Web Services*

# SOAP Faults

- In case an error occurs during the processing of a SOAP message by the server, the server send back a SOAP fault message to the client
- Faults are dealt with by a specific “fault element” within the SOAP body



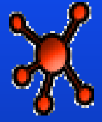


Web Services

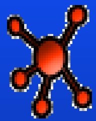
# Fault Message: Example

```
...
<SOAP-ENV:Body>
  <SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Server</faultcode>
    <faultstring >Server Error</faultstring>
    <faultactor>/soap/servlet/rpcrouter</faultactor>
    <detail>
      <e:myfaultdetails xmlns:e="urn:BookService">
      <message>Book out of print</message>
      <errorcode>4711</errorcode>
      </e:myfaultdetails>
    </detail>
  </SOAP-ENV:Fault>
</SOAP-ENV:Body>
...
```

mandatory
mandatory
optional
optional

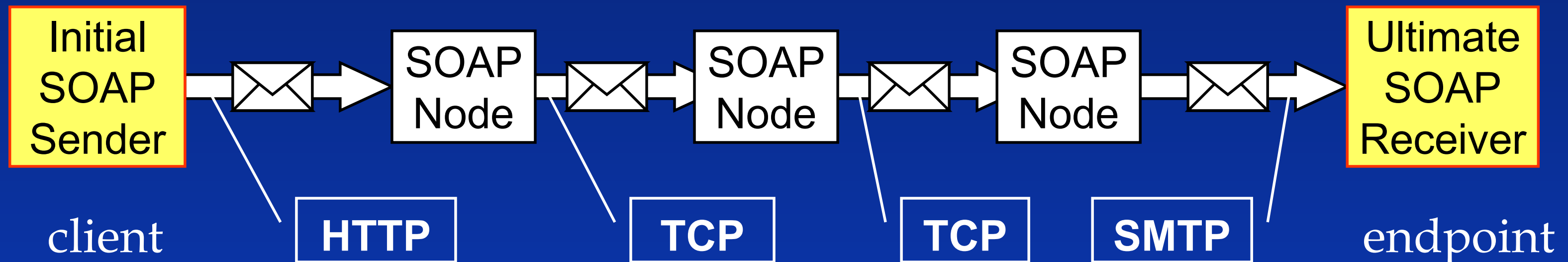


# SOAP Processing Model

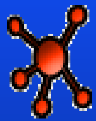


Web Services

# SOAP Processing Model (I)




- Three kinds of SOAP nodes
  - Initial sender, an intermediary, or ultimate receiver
- When processing a message,
  a SOAP node assumes one or more roles
  - Roles determine how headers are processed
- A node first processes mandatory headers (mustUnderstand="1"), then others



Web Services

# SOAP Processing Model (II)

Example:

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
     <wsrp:path xmlns:wsrp="http://schemas.xmlsoap.org/rp"
        soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
        soap:mustUnderstand="1" > ...
```

- Fault generated if node does not understand header
- Successful processing of a header removes it from the message
- Can reinsert the header, but now treated as relationship between the intermediary node and the downstream node
- Ultimate receiver also responsible for processing the Body element