ID2208 Programming Web Services

Simple Object Access Protocol (SOAP)

Mihhail Matskin:
http://people.kth.se/~misha/ID2208/index

Spring 2016

Content

SOAP

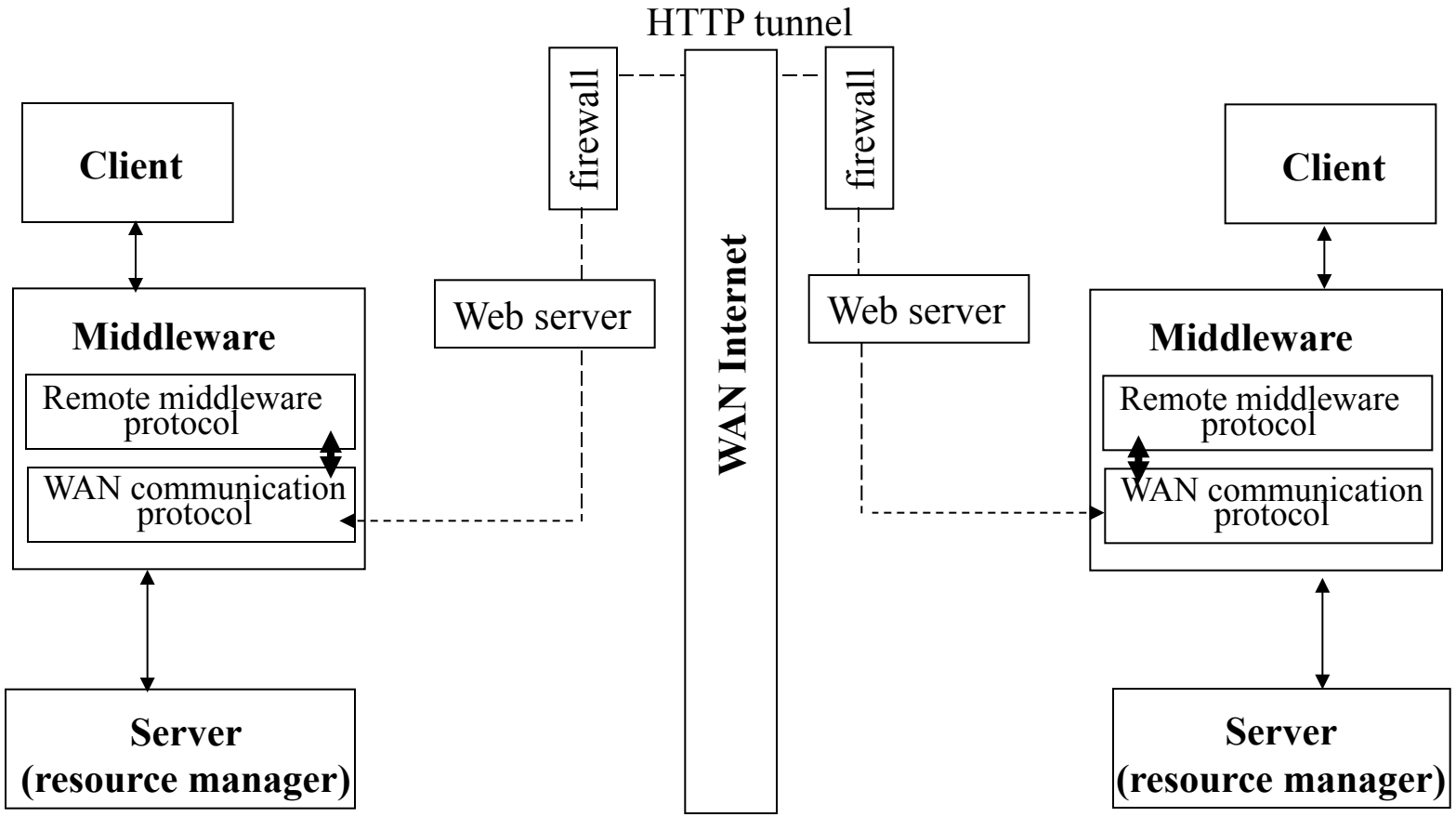
- Introduction
- Architecture
- Types
- Intermediaries
- Data processing
- Messaging

This lecture reference

Text-book **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition**

Chapter 3

B2B interaction using tunneling



Adopted from Alomso et al.

Understanding SOAP

- XML's data portability allows Web services to communicate with applications that run on disparate computing platforms.
- To achieve XML-based interaction, developers need to establish a standard transport and data-exchange framework
- The nature of XML requires that this should be interoperability framework

RPCs and Middleware

RemoteBooking.java

```
package com.psol.resourceful;
import java.util.Date;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface RemoteBooking extends Remote {
   public Resource[] getAllResources()
        throws RemoteException;
   public Resource[] getFreeResourcesOn(Date start,
                                        Date end)
        throws RemoteException;
   public void bookResource(int resource,
                        Date start, Date end, String email)
        throws RemoteException;
}
```

XML-RPC

```
POST /xmlrpc HTTP/1.0
User-Agent: Handson (Win98)
Host: joker.psol.com
Content-Type: text/xml
Content-length: 468
<?xml version="1.0"?>
<methodCall>
   <methodName>
        com.psol.resourceful.BookingService.getFreeResourcesOn
   </methodName>
   <params>
        <param>
          <value>
            <dateTime.iso8601>2001-01-15T00:00:00</dateTime.iso8601>
          </value>
        </param>
        <param>
          <value>
            <dateTime.iso8601>2001-01-17T00:00:00</dateTime.iso8601>
          </value>
        </param>
   </params>
</methodCall>
```

From R.J.Brunner et al. Java Web Services Unleashed. Sams publishing 2002

XML-RPC

```
HTTP/1.0 200 OK
Content-Length: 485
Content-Type: text/xml
Server: Jetty/3.1.4 (Windows 98 4.10 x86)
<?xml version="1.0"?>
<methodResponse>
   <params>
        <param>
            <value>
                <array>
                    <data>
                        <value>
                                <string>Meeting room 1</string>
                        </value>
                        <value>
                                <string>Meeting room 2</string>
                        </value>
                        <value>
                                <string>Board room</string>
                        </value>
                    </data>
                </array>
            </value>
        </param>
   </params>
</methodResponse>
```

From R.J.Brunner et al. Java Web Services Unleashed. Sams publishing 2002

From XML-RPC to SOAP

- XML-RPC is simple and effective
- There's no clean mechanism to pass XML documents themselves in an XML-RPC request or response
- There's no solution that enables programmers to extend the request or response format
- XML-RPC is not fully aligned with the latest XML standardization
- Datatyping in XML-RPC is weak
- RPC is not the only way for application interoperation

A SOAP Request

```
POST /soap/servlet/rpcrouter
HTTP/1.0 Host: joker.psol.com
Content-Type: text/xml; charset=utf-8
Content-Length: 569
SOAPAction: "http://www.psol.com/2001/soapaction"
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <SOAP-ENV:Body>
  <ns1:getFreeResourcesOn  xmlns:ns1="http://www.psol.com/2001/
   resourceful" SOAP-ENV:encodingStyle="http://
   schemas.xmlsoap.org/soap/encoding/">
   <start xsi:type="xsd:timeInstant">2001-01-15T00:00:00Z</start>
   <end xsi:type="xsd:timeInstant">2001-01-17T00:00:00Z</end>
  </ns1:getFreeResourcesOn>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

From R.J.Brunner et al. Java Web Services Unleashed. Sams publishing 2002

A SOAP Response

```
HTTP/1.0 200 OK
Server: Jetty/3.1.4 (Windows 98 4.10 x86)
Servlet-Engine: Jetty/3.1 (JSP 1.1; Servlet 2.2; java 1.3.0)
Content-Type: text/xml; charset=utf-8
Content-Length: 704
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <env:Body>
  <ns1:getFreeResourcesOnResponse xmlns:ns1="http://www.psol.com/
   2001/resourceful" env:encodingStyle="http://
   schemas.xmlsoap.org/soap/encoding/">
   <return xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns2:Array" ns2:arrayType="ns1:String[3]">
        <item xsi:type="xsd:string">Meeting room 1</item>
        <item xsi:type="xsd:string">Meeting room 2</item>
        <item xsi:type="xsd:string">Board room</item>
   </return>
  </ns1:getFreeResourcesOnResponse>
 </env:Body>
</env:Envelope>
```

From R.J.Brunner et al. Java Web Services Unleashed. Sams publishing 2002

What is SOAP?

- SOAP provides a mechanism for defining the unit of communication
- SOAP provides a processing model
- SOAP provides a mechanism for error handling
- SOAP provides an extensibility mechanism
- SOAP provides a flexible mechanism for data representation
- SOAP provides a convention for representing RPCs and responses as SOAP messages
- SOAP support a Document centric approach for documents exchange
- SOAP provides a binding mechanism for SOAP messages to HTTP

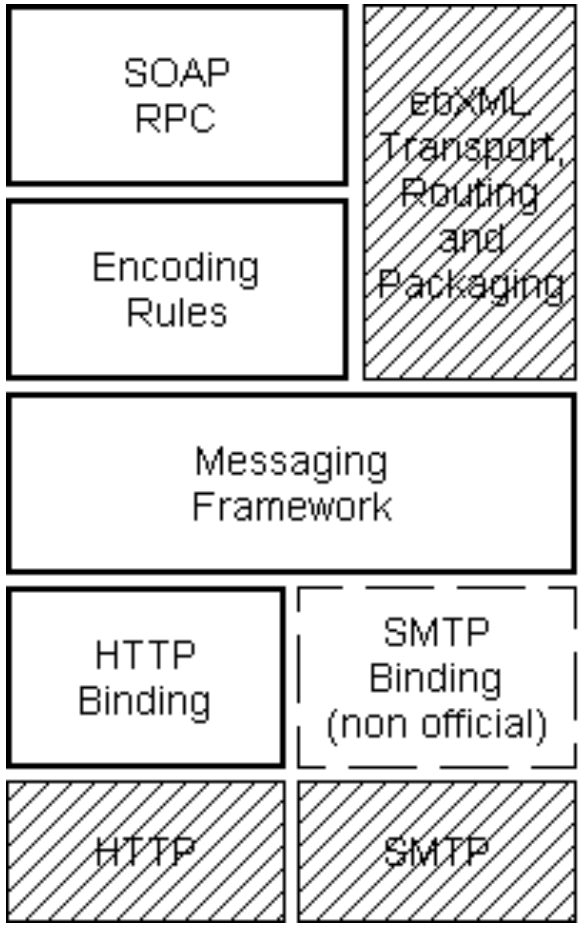
Simple Object Access Protocol (SOAP)

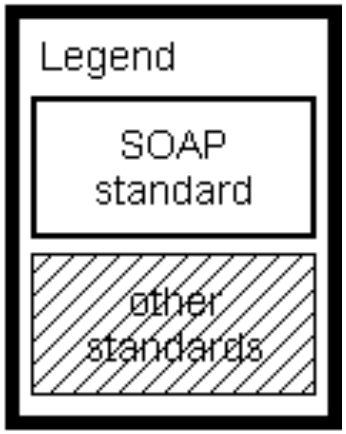
- SOAP specifies a messaging framework (XML documents = SOAP messages)
- SOAP messages encapsulate information transmitted to and from a Web service
- SOAP messages don't provide programming instructions – they rather specify which operation to invoke
- SOAP supports extensibility, attachments, security, routing information and transactions

SOAP architecture

- SOAP envelope specification describes the format of SOAP message
- Data encode rules - Set of rules that encode data types – set of conventions for encoding data
- RPC conventions - execution of SOAP messages via remote procedure call
- SOAP binding framework – defines the protocol through which SOAP messages are transmitted to applications

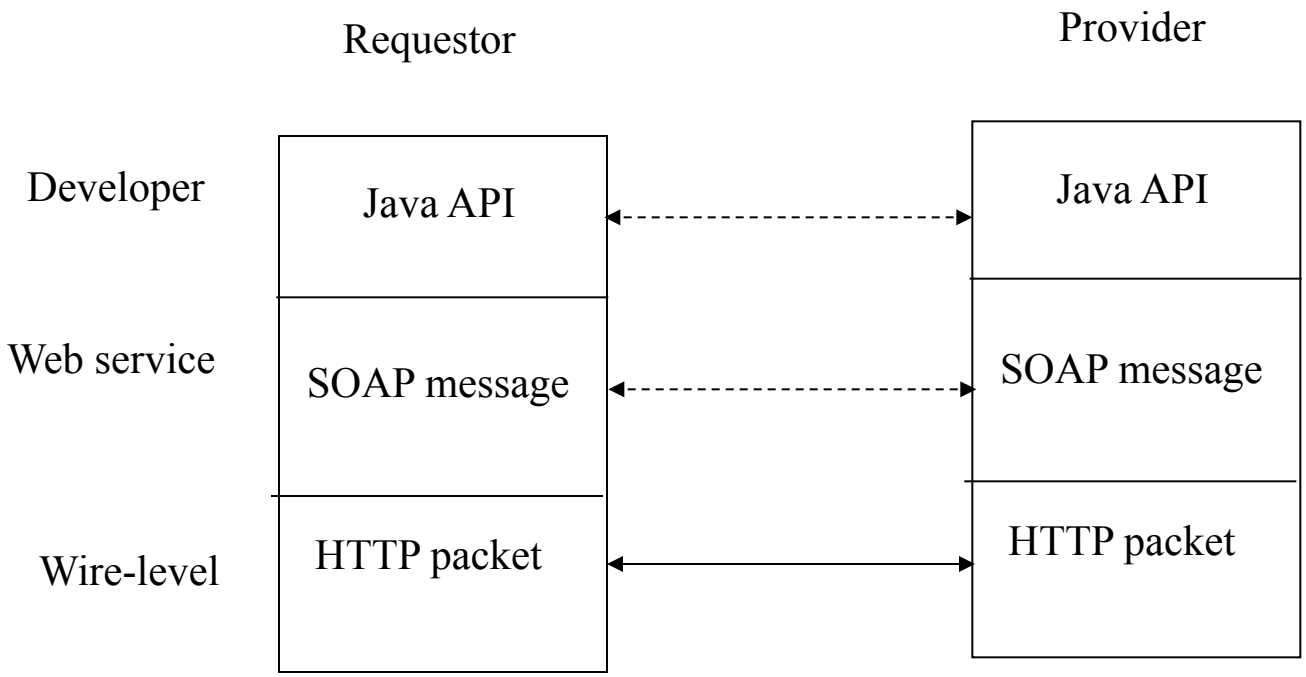
SOAP Architecture





From R.J.Brunner et al. Java Web Services Unleashed. Sams publishing 2002

SOAP Architecture



Inventory Check problem (inventory database)

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product>
      <sku>947-TI</sku>
      <name>Titanium Glider</name>
      <type>skateboard</type>
      <desc>Street-style titanium skateboard.</desc>
      <price>129.00</price>
      <inStock>36</inStock>
  </product>
. . .
</products>
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Inventory Check problem (Product Database Class)

```
public class ProductDB
{
   private Product[] products;
   public ProductDB(Document doc) throws Exception
   {
        // Load product information }
   public Product getBySKU(String sku)
   {
        Product[] list = getProducts();
        for ( int i = 0 ; i < list.length ; i++ )
               if (sku.equals(list[i].getSKU()))
                                          return( list[i]);
        return( null );
   }
   public Product[] getProducts()
   {
        return products;
   }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Inventory Check Web Service (service provider view)

```
import …;
/** Inventory check Web service */
public class InventoryCheck implements STConstants
{
   /* Checks inventory availability given a product SKU and
    * a desired product quantity.
    *
    * @param sku product SKU
    * @param quantity quantity desired
    * @return true|false based on product availability
    * @exception Exception most likely a problem accessing DB
    */
   public static boolean doCheck(String sku, int quantity)
        throws Exception
   {
        ProductDB db = ProductDB.getCurrentDB(DB);
        Product prod = db.getBySKU(sku);
        return (prod != null && prod.getNumInStock() >= quantity);
   }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Inventory Check Web Service (service requestor view)

```
package …;
import org.apache.axis.client.ServiceClient;
// Inventory check web service client
public class InventoryCheckClient
{
   // Service URL
        private String url;
   // Point a client at a given service URL
        public InventoryCheckClient(String targetUrl)
        {
                url = targetUrl;
        }
   // Invoke the inventory check web service

   public boolean doCheck(String sku, int quantity) throws Exception
   {
        Call call = new Call(url);
        Boolean result = (Boolean) call.invoke("", "doCheck", new
                          Object[] { sku, new Integer(quantity) } );
        return result.booleanValue();
   }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

SOAP Request

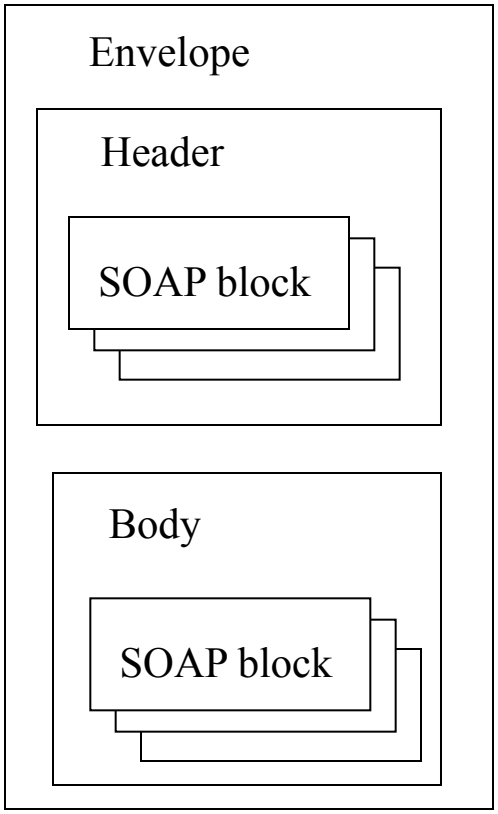
```
POST /axis/InventoryCheck.jws HTTP/1.0
Content-Type: application/soap+xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <doCheck soapenv:encodingStyle=
                "http://www.w3.org/2003/05/soap-encoding" >
        <arg0 xsi:type="soapenc:string“ xmlns:soapenc=
                   "http://schemas.xmlsoap.org/soap/encoding/" >
                947-TI
        </arg0>
        <arg1 xsi:type="soapenc:int“ xmlns:soapenc=
                   "http://schemas.xmlsoap.org/soap/encoding/" >
                1
        </arg1>
   </doCheck>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response

```
HTTP/1.0 200 OK
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope/"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <doCheckResponse soapenv:encodingStyle=
              "http://www.w3.org/2003/05/soap-encoding" >
     <rpc:result xmlns:rpc= "http://www.w3.org/2003/05/soap-rpc">
              return
     </result>
     <return xsi:type="xsd:boolean">true</return>
   </doCheckResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

SOAP:Physical layout of a message



SOAP Elements

- **`env:Envelope`** is the root of the SOAP request.
- **`env:Header`** contains auxiliary information. The header is optional.
- **`env:Body`** contains the main information in one or more SOAP blocks.
- **`env:Fault`** is a special block that indicates protocol-level errors. If present, it must appear in the body.

SOAP versioning

- No explicit protocol version in the Envelop
- SOAP exploits capabilities of XML namespace
- Defines the protocol version as URI of the SOAP envelope name space
  - **`http://schemas.xmlsoap.org/soap/envelope/`**
  - **`http://www.w3.org/2003/05/soap-envelope/`**

Versions

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope/">
   <env:Header>
     <env:Upgrade>
       <env:SupportedEnvelope qname="ns1:Envelope“
         xmlns:ns1="http://www.w3.org/2003/05/soap-envelope"/>
       <env:SupportedEnvelope qname="ns2:Envelope“
        xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope"/>
     </env:Upgrade>
   </env:Header>
   <env:Body>
      <env:Fault>
         <env:Code>
              <env:Value>env:VersionMismatch</env:Value>
         </env:Code>
         <env:Reason>
              <env:Text> Versions Mismatch</env:Text>
         </env:Reason>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

SOAP message (Header)

- Header – offers a flexible framework for specifying additional application level elements, for example:
  - Authentication (digital signature)
  - Payment authorization (account number for pay-per-use SOAP service)
  - Transaction management
  - Message parsing instructions
  - Routing information
  - …
- The details of Header element are open ended. Some attributes can be pre-defined
  - **`actor/role`** attribute
  - **`MustUnderstand`** attribute

Headers

```
<env:Envelope xmlns:env=
       "http://www.w3.org/2003/05/soap-envelope/"/>
  <env:Header>
    <t:Transaction xmlns:t="some-URI“ env:mustUnderstand="1">
              12345
    </t:Transaction>
    <p:Priority xmlns:p="some-Other-URI">
         <ReallyVeryHigh/>
    </p:Priority>
    <notary:token xmlns:notary=“http://notaries-r-us.com”>
              XQ34Z-4G5
    <notary:token>
  </env:Header>
  <env:Body>
       ...
   </env:Body>
</env:Envelope>
```

SOAP Message

```
POST /axis/InventoryCheck HTTP/1.0
Content-Type: application/soap+xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:env="http://www.w3.org/2003/05/soap-envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Header>
   <e:EMail xmlns:e="http://www.skatestown.com/ns/email">
        confirm@partners.com
   </e:EMail>
 </env:Header>
 <env:Body>
   <doCheck env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <arg0 xsi:type="soapenc:string"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">947-TI
    </arg0>
    <arg1 xsi:type="soapenc:int"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">1
    </arg1>
   </doCheck>
 </soapenv:Body>
</soapenv:Envelope>
```

Inventory Check Web Service (service provider view)

```
import …;
/** Inventory check Web service */
public class InventoryCheck implements STConstants
{
   /* Checks inventory availability given a product SKU and
    * a desired product quantity.
    *
    * @param sku product SKU
    * @param quantity quantity desired
    * @return true|false based on product availability
    * @exception Exception most likely a problem accessing DB
    */
   public static boolean doCheck(String sku, int quantity)
        throws Exception
   {
        ProductDB db = ProductDB.getCurrentDB(DB);
        Product prod = db.getBySKU(sku);
        return (prod != null && prod.getNumInStock() >= quantity);
   }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Inventory Check Web Service (service requestor view)

```
package …;
import org.apache.axis.client.ServiceClient;
// Inventory check web service client
public class InventoryCheckClient
{
   // Service URL
        private String url;
   // Point a client at a given service URL
        public InventoryCheckClient(String targetUrl)
        {
                url = targetUrl;
        }
   // Invoke the inventory check web service

   public boolean doCheck(String sku, int quantity) throws Exception
   {
        Call call = new Call(url);
        Boolean result = (Boolean) call.invoke("", "doCheck", new
                          Object[] { sku, new Integer(quantity) } );
        return result.booleanValue();
   }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Service requestor view

```
package …;
import …;
/* * Inventory check web service client */
public class InventoryCheckClient {
 /** * Service URL */
    String url;
/** * Email address to send confirmations to */
    String email;
/** * Point a client at a given service URL */
    public InventoryCheckClient(String url, String email) {
          this.url = url;
          this.email = email;
}
/** * Invoke the inventory check web service */
public boolean doCheck(String sku, int quantity) throws Exception {
    // Build the email header DOM element
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc = builder.newDocument();
    Element emailElem = doc.createElementNS( "http://www.skatestown.com/ns/email", "EMail");
    emailElem.appendChild(doc.createTextNode(email));
    // Build the RPC request SOAP message
    SOAPEnvelope reqEnv = new SOAPEnvelope();
    reqEnv.addHeader(new SOAPHeader(emailElem));
    Object[] params = new Object[]{ sku, new Integer(quantity), };
    reqEnv.addBodyElement(new RPCElement("", "doCheck", params));
    // Invoke the inventory check web service
    ServiceClient call = new ServiceClient(url);
    SOAPEnvelope respEnv = call.invoke(reqEnv);
    // Retrieve the response
    RPCElement respRPC = (RPCElement)respEnv.getFirstBody();
    RPCParam result = (RPCParam)respRPC.getParams().get(0);
    return ((Boolean)result.getValue()).booleanValue(); }
}
```

From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

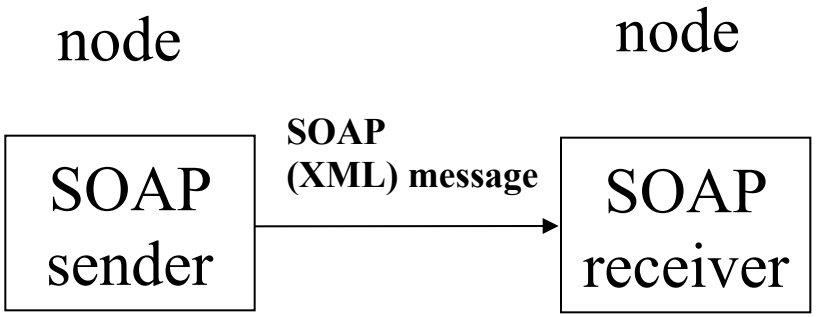
Service Provider View (header processing)

```
package com.skatestown.services;
import …;
/** * EMail header handler */
public class EMailHandler extends BasicHandler {
/** * Utility method to retrieve RPC parameters from a SOAP message. */
 private Object getParam(Vector params, int index){return ((RPCParam)params.get(index)).getValue();}
/**Looks for the EMail header and sends an email confirmation message based on the inventory check */
 public void invoke(MessageContext msgContext) throws AxisFault {
  try {
   // Attempt to retrieve EMail header
   Message reqMsg = msgContext.getRequestMessage();
   SOAPEnvelope reqEnv = reqMsg.getAsSOAPEnvelope();
   SOAPHeader header = reqEnv.getHeaderByName("http://www.skatestown.com/ns/email", "EMail" );
   if (header != null) {
    // Mark the header as having been processed
    header.setProcessed(true);
    // Get email address in header
    String email = (String)header.getValueAsType(SOAPTypeMappingRegistry.XSD_STRING);
    // Retrieve request parameters: SKU & quantity
    RPCElement reqRPC = (RPCElement)reqEnv.getFirstBody();
    Vector params = reqRPC.getParams();
    String sku = (String)getParam(params, 0);
    Integer quantity = (Integer)getParam(params, 1);
    // Retrieve inventory check result
    Message respMsg = msgContext.getResponseMessage();
    SOAPEnvelope respEnv = respMsg.getAsSOAPEnvelope();
    RPCElement respRPC = (RPCElement)respEnv.getFirstBody();
    Boolean result = (Boolean)getParam( respRPC.getParams(), 0);
    // Send confirmation email
    EmailConfirmation ec = new EmailConfirmation(BookUtil.getResourcePath(msgContext,
                                                 "/resources/email.log"));
    ec.send(email, sku, quantity.intValue(), result.booleanValue());
   }
  }
  catch(Exception e)
   { throw new AxisFault(e); }
 }
}
```

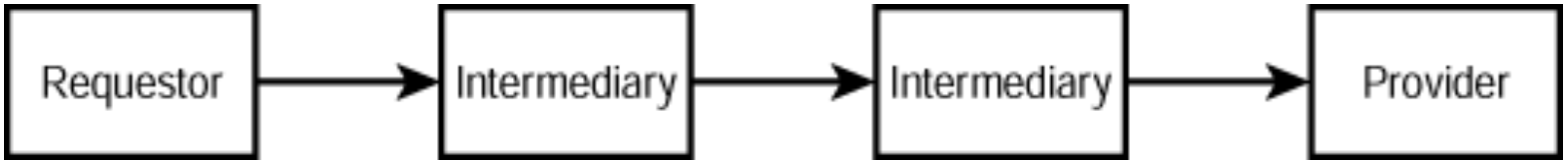
From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

SOAP (message exchange model)

- SOAP node – an application component that understands SOAP
- Message sender
- Message receiver
- SOAP actor/role attribute in header (identified by name – URI)



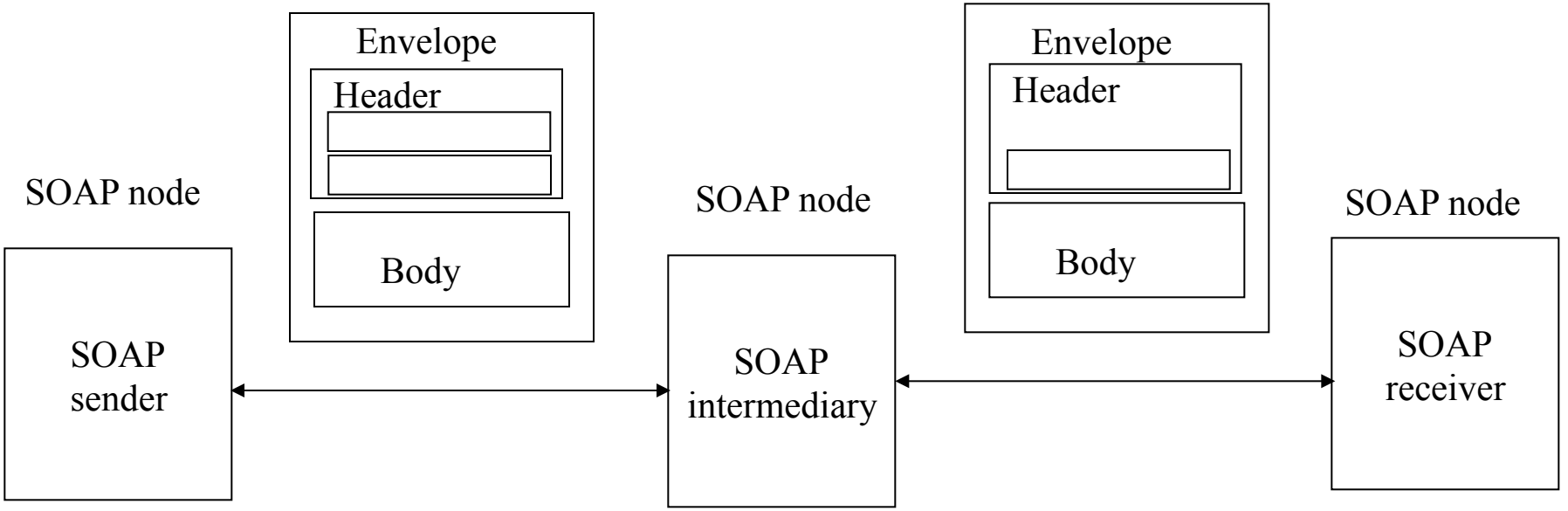
SOAP Intermediaries



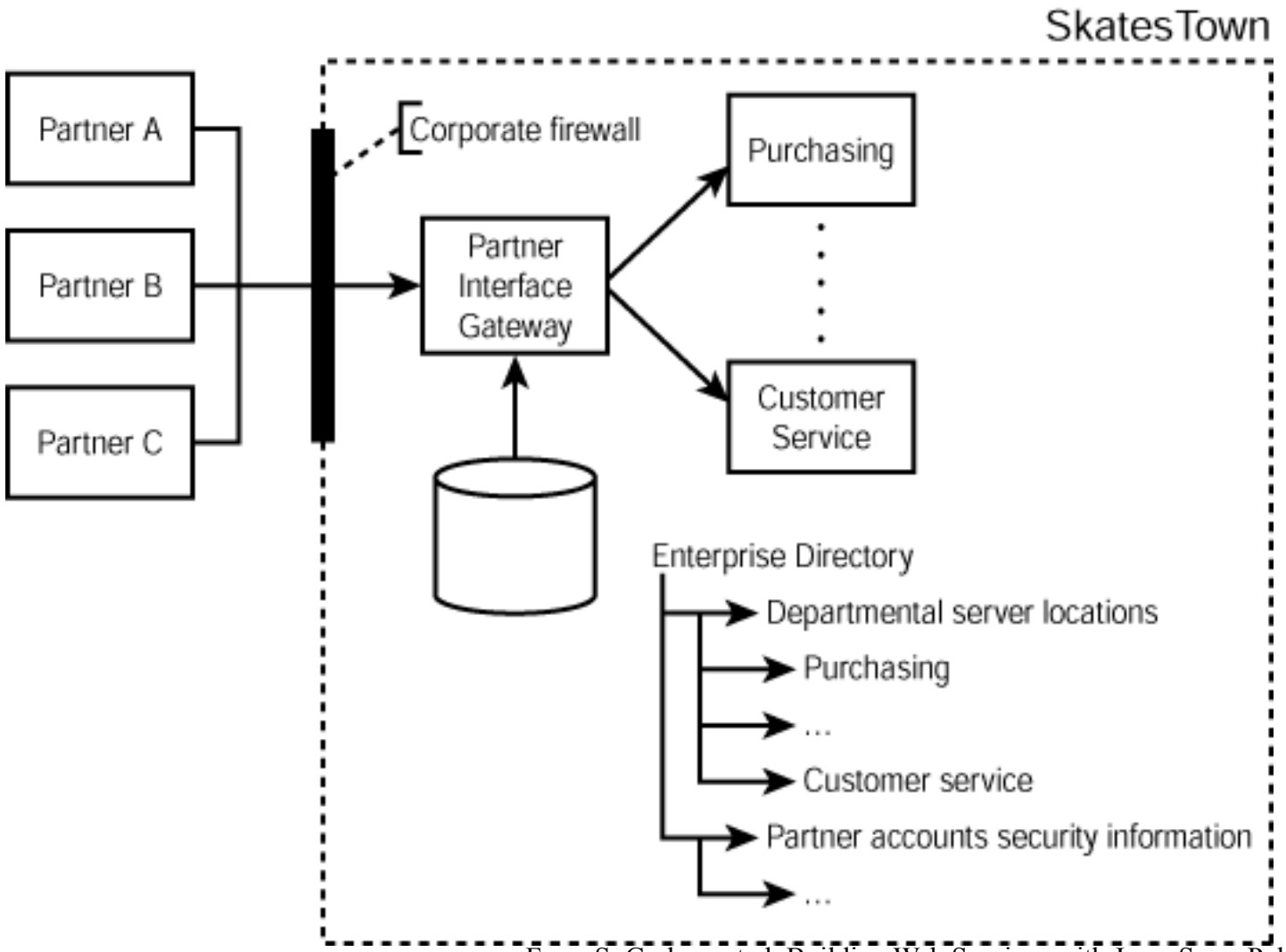
- Intermediary can be transparent or explicit and forwarding or active

- All headers elements can have a **`role`** attribute
  - next
  - ultimateReceiver
  - none

Removal of headers by intermediaries



SkatesTown's overall B2B integration architecture



From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Example message

```
POST /axis/InventoryCheck HTTP/1.0
Content-Type: application/soap+xml; charset=utf-8
HOST: partnergateway.skatetown.com
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:env="http://www.w3.org/2003/05/soap-envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Header>
  <td:TargetDepartment xmlns:td="http://www.skatestown.com/ns/partnergateway"
      env:role="urn:X-SkatesTown:PartnerGateway" env:mustUnderstand="1">
                        Purchasing
  </td:TargetDepartment>
  <ai:AuthenticationInformation xmlns:ai="http://www.skatestown.com/ns/security"
      env:role="urn:X-SkatesTown:PartnerGateway" env:mustUnderstand="1">
        <username>PartnerA</username>
        <password>LongLiveSOAP</password>
  </ai:AuthenticationInformation>
 </env:Header>
 <env:Body>
   <doCheck soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding" >
        <arg0 xsi:type="soapenc:string"
           xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"> 947-TI
        </arg0>
        <arg1 xsi:type="soapenc:int"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" >1</arg1>
        </doCheck>
   </env:Body>
</env:Envelope>
```

Forward message

```
POST /axis/InventoryCheck HTTP/1.0
HOST: purchasing.skatetown.com
Content-Type: application/soap+xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:env="http://www.w3.org/2003/05/soap-envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Header>
  <cc:ClientCredentials env:mustUnderstand="1"
   xmlns:cc="http://schemas.security.org/soap/security">
                <ClientID>/External/Partners/PartnerA</ClientID>
  </cc:ClientCredentials>
 </env:Header>
 <env:Body>
  <doCheck soapenv:encodingStyle="http://www.w3.org/2003/05/soap-
   encoding”>
   <arg0 xsi:type="soapenc:string“
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">947-TI
   </arg0>
   <arg1 xsi:type="soapenc:int“       xmlns:soapenc="http://
   schemas.xmlsoap.org/soap/encoding/">1
   </arg1>
  </doCheck>
 </env:Body>
</env:Envelope>
```

SOAP Processing Model

1. Determine the set of roles in which the node is to act
2. Identify all header blocks targeted at the node that are mandatory
3. If one or more identified mandatory header blocks are not understood the a fault with code set to **`env:mustUnderstood`** is generated and processing is stopped
4. Process all mandatory headers targeted at the node (in case of ultimate receiver also the body)
5. Relay the message if it is intermediary and the exchange pattern and results of processing require that to be done

Body

- Describes the purpose of SOAP message
- Contains data and instructions intended for the receiving application (a typical uses of Body element include RPC requests and responses)
- Surrounds information that is core to the SOAP message.
- May have entries/bodies with arbitrary XML
- Certain conventions govern the format of SOAP body

SOAP Fault message

```
<env:Envelope xmlns:st="http://www.skatestown.com/ws"
        xmlns:env="http://www.w3.org/2003/05/soap-envelope/">
   <env:Header>
        <st:PublicServiceAnnouncement>
           Skatestown's Web services will be unavailable after 5PM
        </st:PublicServiceAnnouncement>
   </env:Header>
   <env:Body>
      <env:Fault>
          <env:Code>
             <env:Value>env:Sender</env:Value>
             <env:Subcode>
                   <env:Value>st:InvalidPurchaseOrder</env:Code>
             </env:Subcode>
          </env:Code>
          <env:Reason>
            <env:Text xml:lang="en_US">
                  Your purchase order did not validate
            </env:Text>
          <env:Detail>
            <st:LineNumber>9</st:LineNumber>
            <st:ColumnNumber>24</st:ColumnNumber>
          </env:Detail>
     </env:Fault>
   </env:Body>
</env:Envelope>
```

SOAP Fault Codes

- **Sender** – the problem was caused by incorrect or missing data from the sender
- **Receiver** – the problem because of something happens on receiver side
- **VersionMismatch** indicates the node does not recognize the version of SOAP used.
- **MustUnderstand** indicates the node does not recognize a block flagged with **`mustUnderstand`**.

Faults

- **Subcodes** – more fine-grain fault codes (they can be nested up to 3 levels)
- **Reason -** contains human-readable description
- **Node** – shows the node which processed the message at the time of fault.
- **Role** – tells which role the faulting node was playing when the fault occurred.

Fault details

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
   envelope">
    <env:Body>
       <env:Fault>
          <env:Code>Sender</env:Code>
           <env:detail>
               <stackTrace>
                java.lang.NullPointerException:
                at com.psol.jws.TroubleMaker(TroubleMaker.java:148)
                at java.lang.Thread.run(Thread.java:484)
               </stackTrace>
           </env:detail>
       </env:Fault>
   </env:Body>
</env:Envelope>
```

Header Faults (NotUnderstood)

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <abc:Extension1 xmlns:abc="http://example.org/2001/06/ext"
        env:mustUnderstand="true"/>
  <def:Extension2 xmlns:def="http://example.org/2001/06/stuff"
        env:mustUnderstand="true"/>
 </env:Header>
…
</env:Envelope>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xml="http://www.w3.org/XML/1998/namespace">
 <env:Header>
   <env:NotUnderstood qname="abc:Extension1"
                      xmlns:abc="http://example.org/2001/06/ext />
 </env:Header>
 <env:Body>
   <env:Fault>
    <env:Code>
     <env:Value>env:mustUnderstand</env:Value>
    </env:Code>
     <env:Reason>
      <env:Text xml:lang="en">Some mandatory SOAP header blocks not understood
      </env:Text>
     </env:Reason>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

Header Faults (Upgrade)

```
…
<env:Header>
   <env:Upgrade>
     <env:SupportedEnvelope qname="ns1:Envelope"
         xmlns:ns1="http://www.w3.org/2003/05/soap-envelope" />
     <env:SupportedEnvelope qname="ns2:Envelope"
         xmlns:ns2="http://www.xmlsoap.org/soap/envelope/" />
   </env:Upgrade>
…
<env:Body>
      <env:Fault>
         <env:Code>
            <env:Value>env:VersionMismatch</env:Value>
         </env:Code>
         <env:Reason>
            <env:Text> Versions Mismatch</env:Text>
         </env:Reason>
      </env:Fault>
</env:Body>
…
```

Encoding data for SOAP

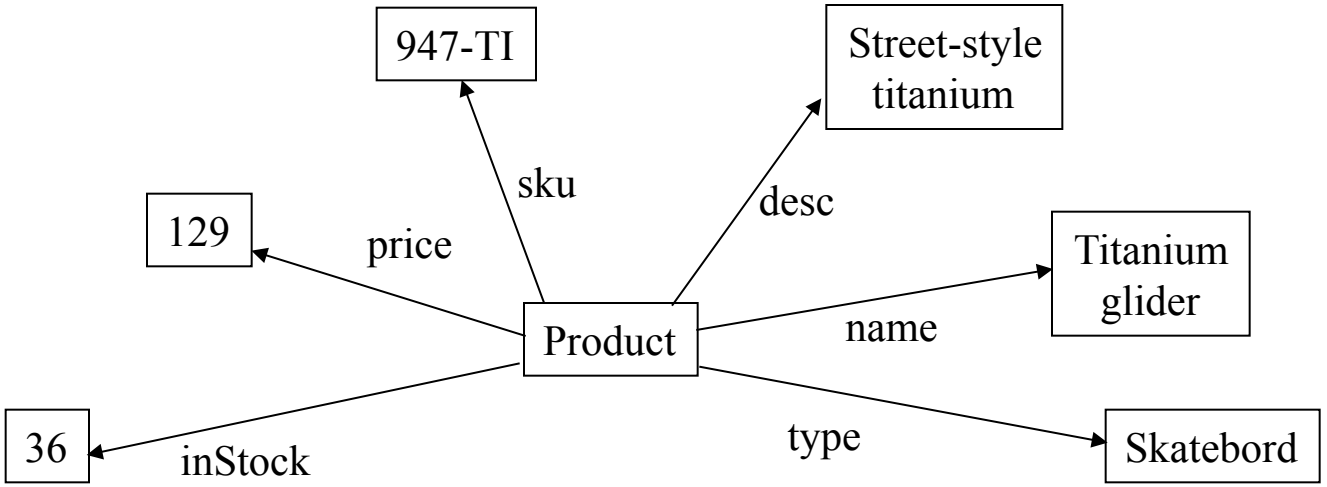
- SOAP does not explicitly map its encoding to programming languages.
- There is no standard SOAP mapping for any language (F/E Java). Therefore, two different implementations of SOAP may produce different encoding for the same objects.
- The most basic rule is that values are always encoded in elements. For example, a name field is encoded as:

  `<name>Board room</name>` but not as:

  `<item name="Board room"/>`

- SOAP uses attributes exclusively to modify the default processing of an element.

Data model and SOAP



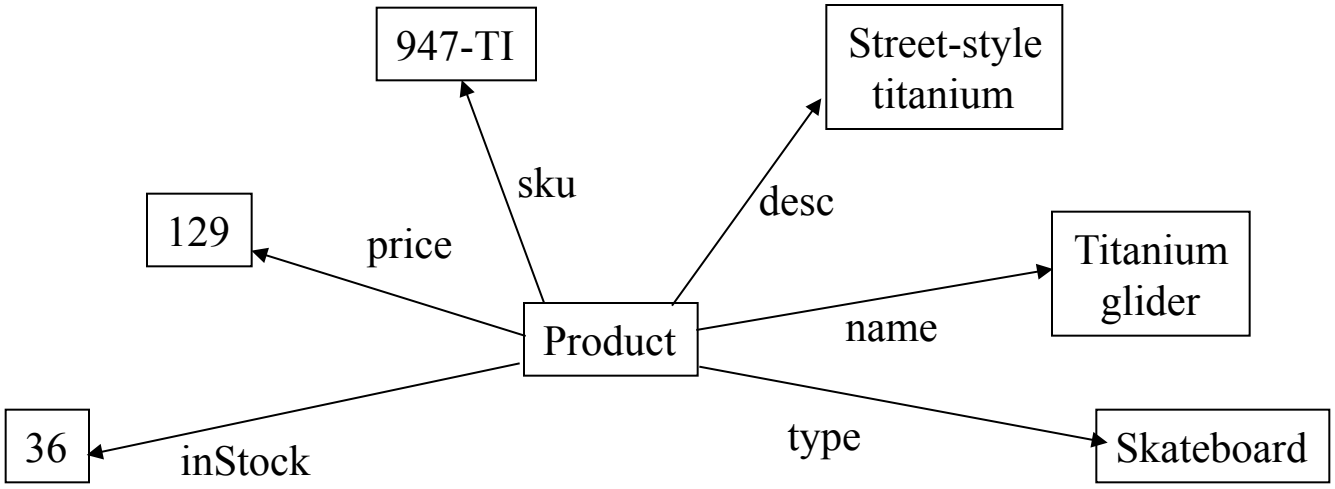

```
Class Product {
   String desc;
   String sku;
   double Price;
   String Name;
   String type;
   int inStock;
}
```

The SOAP Encoding

The encoding style of an element :

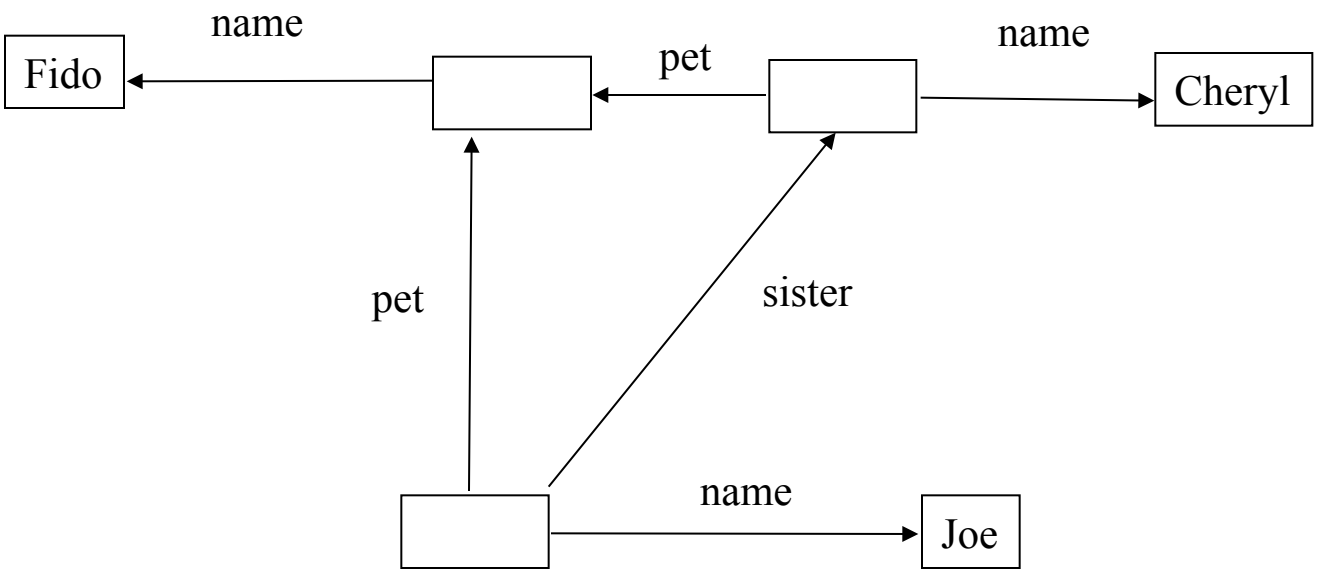
- If an element has the **`encodingStyle`** attribute, then its encoding style is equal to the value of that attribute.
- Otherwise, the encoding style is equal to the encoding style of the closest ancestor element that has the **`encodingStyle`** attribute...
- ...Unless there is no such ancestor, which implies that the element has no specified encoding style.

Data model and SOAP



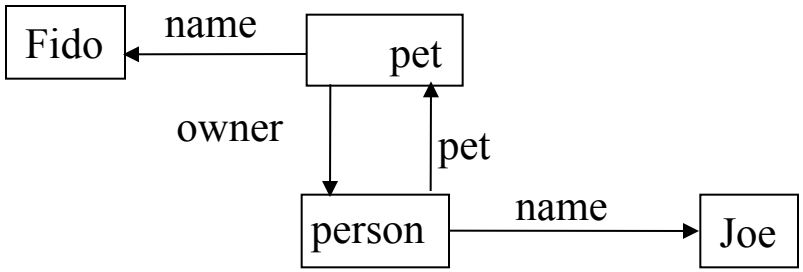

```
<product soapenv:encodingStyle=
        "http://www.w3.org/2003/05/soap-encoding”>
   <sku> 947-TI </sku>
   <name>Titanium glider</name>
   <type>skateboard</type>
   <desc>Street-style titanium</desc>
   <price>129.00</price>
   <inStock>36</inStock>
</product>
```

Data model and SOAP (multirefs)



From S. Graham et al. Building Web Services with Java. Sams Publishing. 2005

Data model and SOAP (multirefs)




```
<person soapenv:encodingStyle=
        "http://www.w3.org/2003/05/soap-encoding"
        id="#1">
   <name>Joe</name>
   <pet id="#2">
        <name>Fido</name>
        <owner ref="#1" />
   </pet>
</person>
```

Compound Types

- A *structure* is a list of elements logically grouped together. The elements are identified by their names. (The element accessor is its name.)
- The *array* is a group of values identified not by their names but by their ordinal positions. (The accessor of the element is the position of the element in the array.)

Arrays

```
<array
     soapenc:itemType="xsd:String“
     soapenc:arraySize=“3”>
  <item>Board room</item>
  <item>Meeting room 1</item>
  <item>Meeting room 2</item>
</array>
```

Arrays

```
<array soapenc:itemType="rs:Resource"
       soapenc:arraySize="3">
   <item>
       <id>1</id>
       <name>Board room</name>
       <description>
              Mid-sized room, quality furniture
       </description>
   </item>
   <item>
       <id>2</id>
       <name>Meeting room 1</name>
       <description>Mid-sized room</description>
   </item>
   <item>
       <id>3</id>
       <name>Meeting room 2</name>
       <description>Small room</description>
    </item>
</array>
```

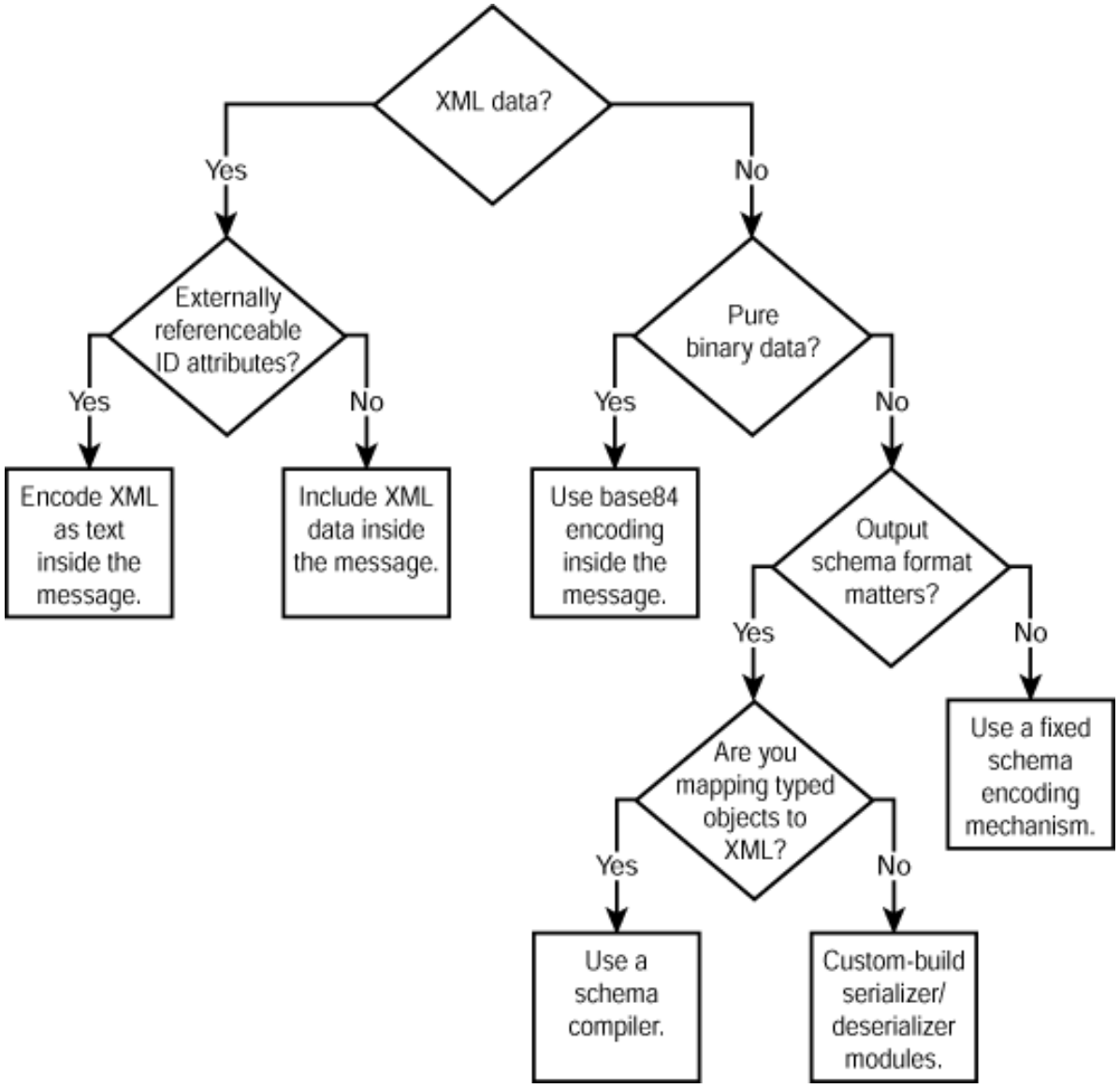
Multidimensional arrays

NW **NE**

SW **SE**

```
<Array soapenc:itemType="xsd:string"
       soapenc:arraySize="2 2">
  <item>NW</item>
  <item>NE</item>
  <item>SW</item>
  <item>SE</item>
</Array>
```

Choosing a Data Encoding



From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

XML data

```
<message id="msg-1">
    A message with attached <a href=“msg-1”>message</a>
  <attachment id="attachment-1">
       <!-- ID conflict right here -->
       <message id="msg-1">
              This is a textually included message.
       </message>
   </attachment>
</message>
```

XML data

```
<message id="msg-1">
   A message with attached
   <a href="id-9137">message</a>
   <attachment id="attachment-1">
        <!-- ID has been changed -->
        <message id="id-9137">
               This is a textually included message.
        </message>
   </attachment>
</message>
```

XML data

```
<message id="msg-1">
   A message with attached &lt;a  href=“msg-1”&gt;
     message&lt;/a&gt; message
   &lt;attachment id="attachment-1"&gt;
  &lt;!-- Message included as text --&gt;
  &lt;message id=“msg-1"&gt;
      This is a textually included message. &lt;/
 message&gt;
  &lt;/attachment&gt;

</message>
```

Binary data

```
<soapenv:Envelope
   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
        . . .
        <Picture xsi:type="xsi:base64binary" imageType="jpg">
              aG93IG5vDyBicm73biBjb3cNCg==…
        </Picture>

   </soapenv:Body>
</soapenv:Envelope>
```

RPC with SOAP

Method

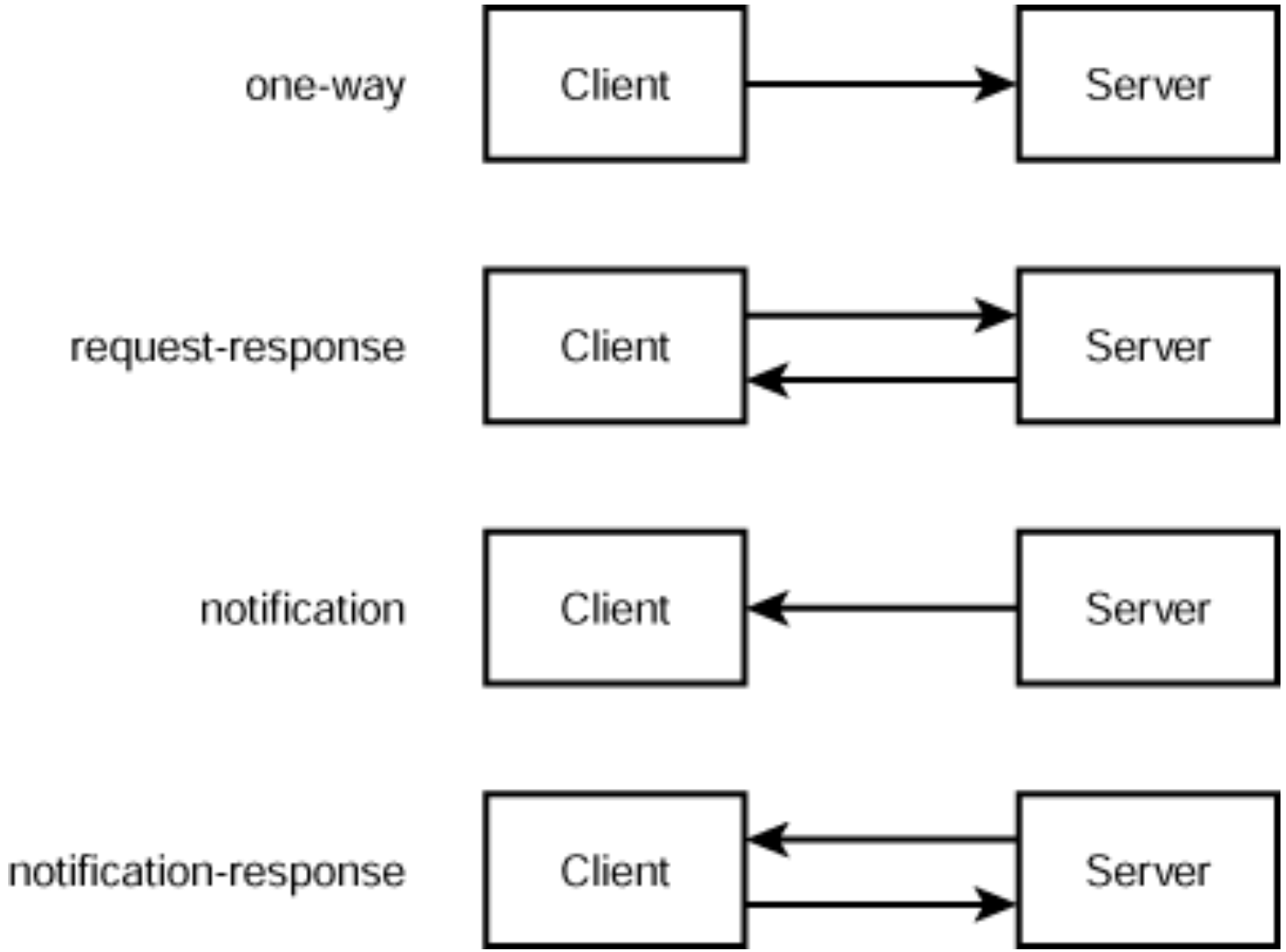
```
boolean doCheck(in String sku, inout  int curRefNum,
                                       out int numInStock)
```

SOAP

```
<!- RPC request body -->
<env:Body>
   <ns1:doCheck soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding“
                xmlns:ns1=“http://www.skatetown.com/”>
        <sku xsi:type=“xsd:string”>947-TI</sku>
        <curRefNum xsi:type=“xsd:int”>1</curRefNum>
   </ns1:doCheck>
</env:Body>

<!- RPC response body -->
<env:Body>
   <ns1:doCheckResponse xmlns:ns1=“http://www.skatetown.com/”
        soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
        xmlns:rpc=“http://www.w3.org/2003/05/soap-rpc”>
        <rpc:result”>return</rpc:result>
        <return xsi:type=”xsd:boolean>true</return>
        <curRefNum xsi:type=“xsd:int”>101</curRefNum >
        <numInStock xsi:type=“xsd:int”>150</numInStock>
   </rpc:doCheckResponse>
</env:Body>
```

Messaging

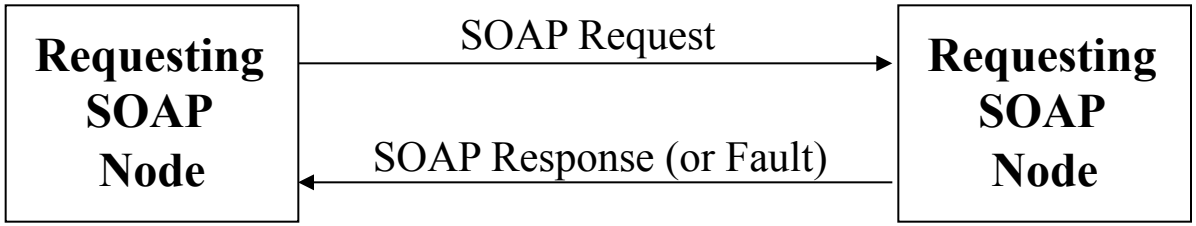


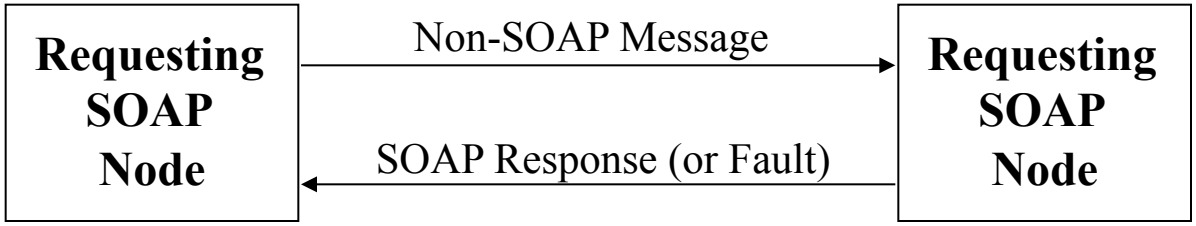
From S. Graham et al. Building Web Services with Java. Sams Publishing. 2002

Features and Properties

- Feature – a semantic that has a name and a specification.
- Features are expressed by
  - Bindings – means for performing functions below the SOAP processing model
  - Modules – means for performing functions using SOAP processing model (headers)
- Properties – pieces of state named with URI which affect the operation of features

Message Exchange Patterns





Message Exchange Patterns

```
<env:Envelope xmlns:env=
                "http://www.w3.org/2003/05/soap-envelope/“>
  <env:Header>
    <!-- Return address -->
    <reqresp:ReplyTo env:mustUnderstand=“true”
        xmlns:reqresp=“http://skatestown.com/requestResponse”>
            <destination>udp://me.com:6666</destination>
    </reqresp:ReplyTo>
    <!-- correlation ID -->
    <reqresp:correlationID env:mustUnderstand=“true”
        xmlns:reqresp=“http://skatestown.com/requestResponse”>
                    1234
    </reqresp:correlationID>
  </env:Header>
…
</env:Envelope>
```

Transport Options (HTTP)

Physical (communication protocol) message

```
POST /LookupCentral HTTP /1.1
Host: www.lookupcentralserver.com
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnn
action= "Directory/LookupPerson"
```

Logical SOAP message

```
<soapenv:Envelope
    xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope /"
    soapenv:encodingStyle="http://www.w3.org/2003/05/soap-encoding /"/>
```

SOAP Headers

```
    <soapenv:Header>
```

In-message context

```
        <a:AuthorizationLevel
            xmlns:a="some-URI">
                <ReallyVeryHigh/>
        </a:AuthorizationLevel>
```

```
    </soapenv:Header>
```

SOAP Body

```
    <soapenv:Body>
        <m:LookupPerson
            xmlns:m="Some-URI">
                <FirstName>Big</FirstName>
                <LastName>Boss</LastName>
        </m:LookupPerson>
    </soapenv:Body>

</soapenv:Envelope>
```

SOAP and HTTP

- Normal SOAP exchange uses HTTP POST operation, however, other semantics can be defined via the Web Method Feature and corresponding property

  `http://www.w3.org/2003/05/soap/features/web-method/method`
- The Feature defines a property which contains one of words GET, POST, PUT, DELETE or other that can be defined.
  - When sending a message HTTP binding uses the specified in the property word instead of POST
- The MIME media type of both HTTP requests and responses must be `application/soap+xml`.
- Successful SOAP message processing must return an HTTP error code in the 200 range.
- In the case of an error processing the SOAP message, the HTTP response code is 400 for soapenv:Sender fault and 500 Internal Server Error otherwise

SOAP Messages with Attachments

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
   type=application/soap+xml;
   start="<claim061400a.xml@claiming-it.com>"
Content-Description: This is the optional message description.
--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<soap:Envelope xmlns:soap=" http://www.w3.org/2003/05/soap-envelope">
   <soap:Body>
      <submitClaim>
..
        <theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
      </submitClaim>
..
   </soap:Body>
</soap:Envelope>
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>
...binary TIFF image...
--MIME_boundary--
```

SOAP and SMTP

- E-mail messages can carry SOAP messages.
- E-mail messages have extensible headers that can be used to transmit context information outside the SOAP message body.
- Both sending and receiving of e-mail messages can be configured to require authentication. E-mail can support one-to-one and one-to-many participant configurations.
- E-mail messaging is buffered and queued with reliable dispatch.
- The Internet e-mail server infrastructure is highly scalable.

Summary

- The evolution of XML protocols from first-generation technologies based on pure XML 1.0 (XML-RPC) to XML Schema and Namespace powered second-generation protocols
- The simple yet flexible design of the SOAP envelope framework, including versioning and vertical extensibility using SOAP headers.
- SOAP intermediaries as the key innovation enabling horizontal extensibility.
- SOAP error handling using SOAP faults. Encoding data using SOAP. Using SOAP for both messaging and RPC applications.
- Using SOAP over multiple protocols.

(REpresentational State Transfer) RESTful Web Services

- Architectural style which consists of clients and servers.
- The RESTful Web services are viewed as resources and can be identified via URLs
- Actions with resources can be done by a limited set of globally defined methods
- While REST, in general, is not limited to HTTP, in practice it is used in the context of HTTP

Adopted from oracle.com

Basic principles

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON), or both.

RESTful Web Service HTTP methods

RESTful resource	GET	PUT	POST	DELETE
Collection URI, such as http://RESTexample.eu/resources/	List the members of some collection with their member URIs	Create a new collection (if not exists yet). Replace the existing collection otherwise.	For example, creates a new entry in the collection. The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI	Delete the entire collection
Element URI, such as http://RESTexample.eu/resources/ID82736	Retrieve a representation of the addressed element	Update the addressed element of the collection or create it with the specified ID.	For example, creates a new subordinate of the element. The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI	Delete the addressed element of the collection.

Adopted from WIKI

RESTful web services and HTTP methods

- GET is used to retrieve data or perform a query on a resource.
- POST is used to create a new resource.
- PUT is used to update existing resources or data.
- DELETE is used to remove a resource or data.

Adopted from oracle.com

Method information and Scope information

- What to do with data? Method information
  - GET, PUT, DELETE, POST for HTTP
- Which part of data set to operate on? Scoping information
  - URI path (not in the entity body)
- `http://flickr.com/photos/tags/penguin`
- `http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=penguin`

RESTful web services

- Non proper usage:
  - `GET /addstudent?name=John HTTP/1.1`
  - `POST /deletestudent?name=John HTTP/1.1`
- A proper usage:
  - `POST /users HTTP/1.1`

```
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
  <student>
        <name>John</name>
  </student>
```

Create PO

```
POST /restfulexample/po/ HTTP/1.0
Accept: */*
Connection: close
Content-Type: text/xml
Content-Length: 618
. . .
<po id="44581" submitted="2004-06-05">
 <billTo id="addr-1">. . . </billTo>
 <shipTo href="addr-1"/>
 <order>
    <item sku="316-BP" quantity="5" unitPrice="49.95">
      <description>Skateboard backpack</description>
    </item>
    . . .
 </order>
 <tax>89.89</tax>
 <shippingAndHandling>200</shippingAndHandling>
 <totalCost>2087.64</totalCost>
</po>
```

Read PO

```
GET /restfulexample/po/73452 HTTP/1.0
Connection: close
Content-Type: text/xml
```

Update PO

```
PUT /restfulexample/po/73452 HTTP/1.0
Connection: close
Content-Type: text/xmlContent-Length: xxx
. . .
<po id="44581" submitted="2004-06-05">
 <billTo id="addr-1">. . . </billTo>
 <shipTo href="addr-1"/>
 <order>
    <item sku="316-BP" quantity="6" unitPrice="49.95">
      <description>Skateboard backpack</description>
    </item>
    . . .
</po>
```

Delete PO

```
DELETE /restfulexample/po/73452 HTTP/1.0
Connection: close
Content-Type: text/xml
Content-Length: 0
. . .
```

Steps in RESTful Web Services requests

- Come up with data that will go into the HTTP request
- Form the data as an HTTP request and send it to the appropriate HTTP server
- Parse the response data (the response code, headers and entity-body) into the data structure the rest of your program needs

When to Use REST?

- Lightweight – not a lot of extra markup
- Easy to build? – no toolkits required
- A caching infrastructure can be used for performance.
- The service producer and service consumer have a mutual understanding of the context and content being passed along.
- Bandwidth is particularly important and needs to be limited.

When to Use SOAP?

- Easy to consume by application
- Need support Transactions, Security, Addressing, Trust, Coordination, and so on
- Service discovery is needed
- A formal contract must be established to describe the interface that the web service offers.
- The architecture must address complex nonfunctional requirements.
- Services need to be composed into a new service.
- The architecture needs to handle asynchronous processing and invocation

Adopted from java.sun.com

Next lecture

- WSDL

  Text-book **Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition**

Chapter 4