SOAP I: Intro and Message Formats

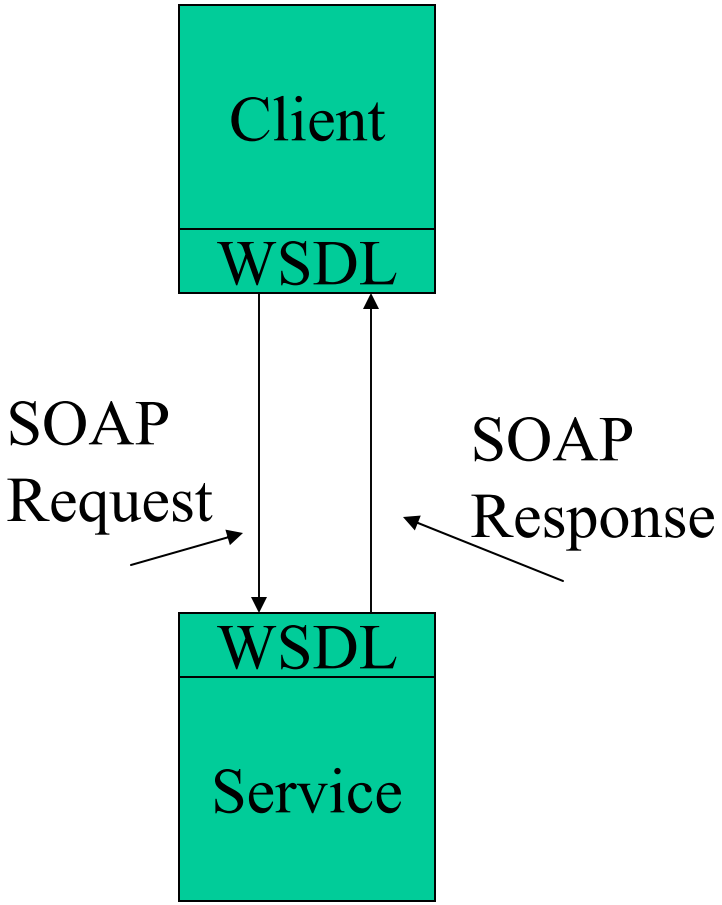
Marlon Pierce, Bryan Carpenter, Geoffrey Fox
Community Grids Lab
Indiana University
mpierce@cs.indiana.edu
http://www.grid2004.org/spring2004

SOAP Primary References

- SOAP is defined by a number of links
  - http://www.w3.org/TR/soap/
- See primarily the “Primer” and “Messaging Framework” links.
- The actual SOAP schema is available from http://www.w3.org/2003/05/soap-envelope/
  - It is pretty small, as these things go.

SOAP and Web Services

- Our previous lectures have looked at WSDL
  - Defines the interfaces for remote services.
  - Provides guidelines for constructing clients to the service.
  - Tells the client how to communicate with the service.
- The actual communications are encoded with SOAP.
  - Transported by HTTP



Defining SOAP Messages

- Given what you have learned about WSDL, imagine it is your job to design the message interchange layer.
  - What are the requirements?
- Note SOAP actually predates WSDL, so this is in reverse order.

Web Service Messaging Infrastructure Requirements?

- Define a message format
  - Define a messaging XML schema
  - Allow the message to contain arbitrary XML from other schemas.
- Keep It Simple
  - Messages may require advanced features like security, reliability, conversational state, etc.
  - KISS, so don't design these but do design a place where this sort of advanced information can go.
- Tell the message originator is something goes wrong.
- Define data encodings
  - That is, you need to tell the message recipient the types of each piece of data.
- Define some RPC conventions that match WSDL
  - Your service will need to process the message, so you need to provide some simple conventions for matching the message content to the WSDL service.
- Decide how to transport the message.
  - Generalize it, since messages may pass through many entities.
- Decide what to do about non-XML payloads (movies, images, arbitrary documents).

SOAP Lecture Parts

- SOAP Messages:
  - Headers and body elements with examples.
- SOAP Encoding:
  - Rules for encoding data.
  - Focus on SOAP for RPC
- SOAP Routing and Processing
- SOAP Over HTTP:
  - How SOAP gets sent over the wire.

SOAP Messaging

SOAP Basics

- SOAP is often thought of as a protocol extension for doing Remote Procedure Calls (RPC) over HTTP.
  - This is how we will use it.
- This is not completely accurate: SOAP is an XML message format for exchanging structured, typed data.
  - It may be used for RPC in client-server applications
  - May be used to send XML documents
  - Also suitable for messaging systems (like JMS) that follow one-to-many (or publish-subscribe) models.
- SOAP is not a transport protocol. You must attach your message to a transport mechanism like HTTP.

What Does SOAP Look Like?

- The next two slides shows examples of SOAP message.
  - It's just XML
- First slide is an example message that might be sent from a client to the echo service.
- Second slide is an example response.
  - I have highlighted the actual message payload.

SOAP Request

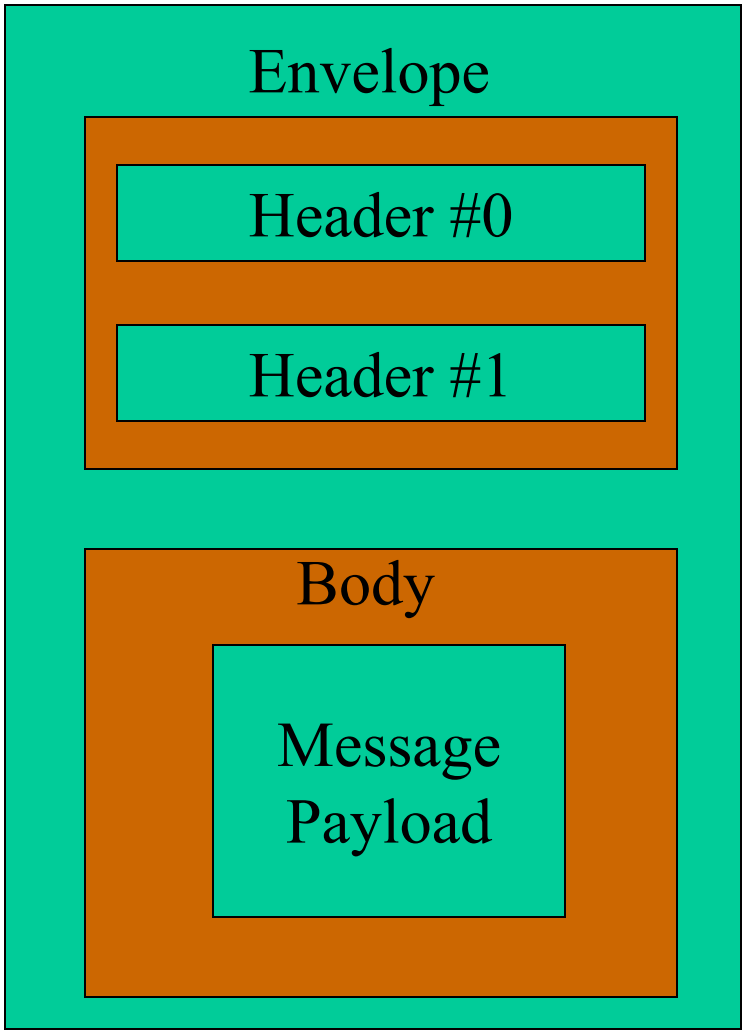
```
<?xml version='1.0' ?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:echo
          soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:ns1="http://.../axis/services/EchoService">
        <in0 xsi:type="xsd:string">Hello World</in0>
    </ns1:echo>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response

```
<?xml version=‘1.0’ ?>
<soapenv:Envelope
          xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
          xmlns:xsd=http://www.w3.org/2001/XMLSchema
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <ns1:echoResponse
        soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        xmlns:ns1="http://../axis/services/echoService">
      <echoReturn xsi:type=“String“> Hello World</echoReturn>
   </ns1:echoResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Structure

- SOAP structure is very simple.
  - 0 or more headers elements
  - 1 body element
  - Envelop that wraps it all.
- Body contains XML payload.
- Headers are structured the same way.
  - Can contain additional payloads of “metadata”
  - Security information, quality of service, etc.



SOAP Schema Notes

- All of this is expressed formally in the SOAP schema.
- XML on the right is taken directly from the SOAP schema.
- This just encodes the previously stated rules.
- Also, note that the SOAP envelope can contain other attributes.
  - <anyAttribute> tag is the wildcard

```
<xs:complexType
    name="Envelope">
 <xs:sequence>
     <xs:element ref="tns:Header"
          minOccurs="0" />
     <xs:element ref="tns:Body"
          minOccurs="1" />
 </xs:sequence>
 <xs:anyAttribute
   namespace="##other"
   processContents="lax" />
</xs:complexType>
```

Options on **<xsd:any/>** (From DBC's Schema Lectures)

- The **<xsd:any/>** element takes the usual optional **maxOccurs**, **minOccurs** attributes.
- Allows a **namespace** attribute taking one of the values:
  - **##any** (the default),
  - **##other** (any namespace except the target namespace),
  - List of namespace names, optionally including either **##targetNamespace** or **##local**.

  Controls what elements the wildcard matches, according to namespace.
- It also allows a **processContents** attribute taking one of the values **strict**, **skip**, **lax** (default **strict**), controlling the extent to which the contents of the matched element are validated.

Lax

- “If the item, or any items among its children if it's an element information item, has a uniquely determined declaration available, it must be ·valid· with respect to that definition.”
- That is, ·validate· where you can, don't worry when you can't.

SOAP Envelop

- The envelop is the root container of the SOAP message.
- Things to put in the envelop:
  - Namespaces you will need.
    - **http://schemas.xmlsoap.org/soap/envelope** is required, so that the recipient knows it has gotten a SOAP message.
    - Others as necessary
  - Encoding rules (optional)
    - Specific rules for deserializing the encoded SOAP data.
    - More later on this.
- Header and body elements.
  - Headers are optional, body is mandatory.
  - Headers come first in the message, but we will look at the body first.

SOAP Headers

- SOAP Body elements contain the primary message contents.
- Headers are really just extension points where you can include elements from other namespaces.
  - i.e., headers can contain arbitrary XML.
- Headers may be processed independently of the body.
- Headers may optionally define encodingStyle.
- Headers may optionally have a “role” attribute
- Header entries may optionally have a “mustUnderstand” attribute.
  - mustUnderstand=1 means the message recipient must process the header element.
  - If mustUnderstand=0 or is missing, the header element is optional.

Header Definition From SOAP Schema

```
<xs:element name="Header" type="tns:Header" />
  <xs:complexType name="Header">
    <xs:annotation>
      <xs:documentation>Elements replacing the wildcard MUST be
          namespace qualified, but can be in the
          targetNamespace</xs:documentation>
    </xs:annotation>
    <xs:sequence>
          <xs:any namespace="##any" processContents="lax"
          minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>
```

Example Uses of Headers

- Security: WS-Security and SAML place additional security information (like digital signatures and public keys) in the header.
- Quality of Service: SOAP headers can be used if we want to negotiate particular qualities of service such as reliable message delivery and transactions.
  - We will look at reliable messaging in detail in a future lecture.
- Session State Support: Many services require several steps and so will require maintenance of session state.
  - Equivalent to cookies in HTTP.
  - Put session identifier in the header.

Example Header from SOAP Primer

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="…"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d
      </m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00
     </m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="…"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
          <n:name>Åke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
```

Explanation of Header Example

- In this particular case, we may imagine an ongoing transaction for making an airline reservation.
  - Involves several steps and messages, so client must remind the server of this state information when sending a message.
  - The actual header content all comes from other namespaces.
- The role and mustUnderstand attributes are from SOAP.

Header Processing

- SOAP messages are allowed to pass through many intermediaries before reaching their destination.
  - Intermediary=some unspecified routing application.
  - The final destination processes the body of the message.
- Headers are allowed to be processed independently of the body.
  - May be processed by intermediaries.
- This allows an intermediary application to determine if it can process the body, provide the required security, session, or reliability requirements, etc.

Header Roles

- SOAP nodes may be assigned role designations.
- SOAP headers then specify which role or roles should process.
- Standard SOAP roles:
  - **None**: SOAP nodes MUST NOT act in this role.
  - **Next:** Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
  - **UltimateReceiver:** The ultimate receiver MUST act in this role.
- In our example, all nodes must process the header entries.

SOAP Body

- Body entries are really just placeholders for XML from some other namespace.
- The body contains the XML message that you are transmitting.
- It may also define encodingStyle, just as the envelop.
- The message format is not specified by SOAP.
  - The <Body></Body> tag pairs are just a way to notify the recipient that the actual XML message is contained therein.
  - The recipient decides what to do with the message.

SOAP Body Element Definition

```
<xs:element name="Body" type="tns:Body" />
<xs:complexType name="Body">
    <xs:sequence>
      <xs:any namespace="##any"
        processContents="lax" minOccurs="0“
        maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other"
      processContents="lax" />
</xs:complexType>
```

SOAP Body Example

```
<soapenv:Body>
  <ns1:echo soapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1=
        "http://.../axis/services/EchoService">
       <in0 xsi:type="xsd:string">Hello
        World</in0>
  </ns1:echo>
</soapenv:Body.
```

Example SOAP Body Details

- The <Body> tag is extended to include elements defined in our Echo Service WSDL schema.
- This particular style is called RPC.
  - Maps WSDL bindings to SOAP body elements.
  - Guidelines will be given in next lecture.
- xsi-type is used to specify that the <in0> element takes a string value.
  - This is data encoding
  - Data encoding rules will also be examined in next lectures.

When Things Go Wrong

- One of the precepts of distributed computing is that things will go wrong in any operational system.
  - Servers will fail, networks will go down, services will change or go away.
- Need a way to communicate failures back to message originators.
  - Consider HTTP faults
- SOAP Provides its own fault communication mechanism.
- These may be in addition to HTTP errors when we use SOAP over HTTP.

- HTTP Error Messages

  403 Forbidden
  404 Not Found
  405 Method Not Allowed
  406 Not Acceptable
  407 Proxy Authentication Required
  408 Request Time-Out
  409 Conflict
  410 Gone
  411 Length Required
  412 Precondition Failed
  413 Request Entity Too Large
  414 Request-URL Too Large
  415 Unsupported Media Type
  500 Server Error
  501 Not Implemented
  502 Bad Gateway
  503 Out of Resources
  504 Gateway Time-Out
  505 HTTP Version not supported

SOAP Fault Scenarios

- HTTP errors will take precedence.
  - Involve message transmission problems.
- SOAP errors occur during the processing of the message.
  - HTTP 500 Internal Server Error
- Faults can occur when
  - You sent an improperly formatted message that the service can't process (an integer instead of a string, for example).
  - There is a SOAP version mismatch
    - You sent SOAP 1.2 and I understand SOAP 1.0
  - You have a "must understand" header that can't be understood.
  - You failed to meet some required quality of service specified by a header.

Sample SOAP Fault From SOAP Primer

```
<env:Body>
    <env:Fault>
        <env:Code>
          <env:Value>env:Sender</env:Value>
          <env:Subcode>
                <env:Value>rpc:BadArguments</env:Value>
          </env:Subcode>
        </env:Code>
        <env:Reason>
                <env:Text xml:lang="en-US">Processing error</env:Text>
        </env:Reason>
        <env:Detail>
                <e:myFaultDetails> ...</e:myFaultDetails>
        </env:Detail>
    </env:Fault>
</env:Body>
```

Fault Structure from SOAP Schema

- Fault messages are included in the <body>.
- <Code> and <Reason> are required.
- <Node>,<Role>, and <Detail> are optional.

```
<xs:element name="Fault" type="tns:Fault"
    />
<xs:complexType name="Fault“
    final="extension">
    <xs:sequence>
      <xs:element name="Code“
        type="tns:faultcode" />
      <xs:element name="Reason"
        type="tns:faultreason" />
      <xs:element name="Node"
        type="xs:anyURI“ minOccurs="0" />
      <xs:element name="Role"
        type="xs:anyURI" minOccurs="0" />
      <xs:element name="Detail“
        type="tns:detail" minOccurs="0" />
    </xs:sequence>
 </xs:complexType>
```

SOAP Fault Codes

- These are one of the required subelements of Faults.
- They must contain one of the standard fault code enumerations (next slide).
- They may also contain subcodes.
  - For more detailed error messages.

```
<xs:complexType
    name="faultcode">
  <xs:sequence>
    <xs:element
      name="Value"
      type="tns:faultcodeEnum" />
    <xs:element name="Subcode"
      type="tns:subcode"
      minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

Enumerating Faults

- Fault codes must contain one of the standard fault messages.
- DataEncodingUnknown: you sent data encoded in some format that I don't understand.
- MustUnderstand: I don't support this header.
- Receiver: message was correct, but receiver could not process for some reason.
- Sender: message was incorrectly formatted, or lacked required additional information
  - Couldn't authenticate you
- VersionMismatch: I don't support your version of SOAP.

```
<xs:simpleType name="faultcodeEnum">
  <xs:restriction base="xs:QName">
    <xs:enumeration
          value="tns:DataEncodingUnkno
    wn" />
    <xs:enumeration
          value="tns:MustUnderstand" />
     <xs:enumeration
          value="tns:Receiver" />
    <xs:enumeration
          value="tns:Sender" />
    <xs:enumeration
          value="tns:VersionMismatch" />
  </xs:restriction>
</xs:simpleType>
```

Fault Subcodes

- Fault codes may contain subcodes that refine the message.
- Unlike Codes, subcodes don't have standard values.
  - Instead, they can take any QName value.
  - This is an extensibility mechanism.
- Subcodes may contain other subcodes.

```
<env:Code>
    <env:Value>env:Sender
    </env:Value>
    <env:Subcode>
        <env:Value>rpc:Bad
                Arguments
        </env:Value>
    </env:Subcode>
</env:Code>
```

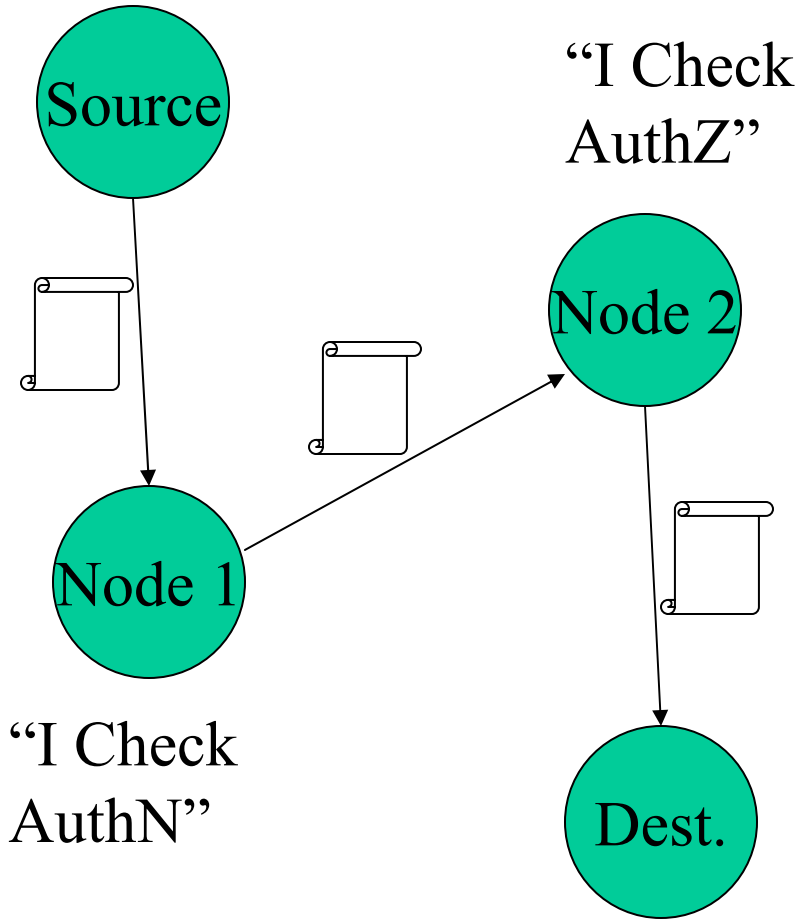
Fault Reasons

- This is intended to provide human readable reasons for the fault.
- The reason is just a simple string determined by the implementer.
  - For Axis, this is the Java exception name.
  - At least, for my version of Axis.
- We must also provide at least one language.

```
<xs:complexType name="faultreason">
  <xs:sequence>
    <xs:element name="Text"
          type="tns:reasontext"
          minOccurs="1"
          maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="reasontext">
  <xs:simpleContent>
  <xs:extension base="xs:string">
    <xs:attribute ref="xml:lang"
          use="required" />
    </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
```

Optional Fault Elements

- Code and Reason are required.
- Node, Role, and Detail are optional
- Node and Role are used in SOAP processing steps that we have lightly covered.
  - SOAP messages may go through several intermediaries.
- Nodes and roles are needed in case a fault occurs in an intermediary.
  - Return the URI of the node and role
- Details will be described.



Fault Detail

- A fault detail is just an extension element.
  - Carries application specific information
- It can contain any number of elements of any type.
- This is intended for the SOAP implementer to put in specific information.
  - You can define your own SOAP fault detail schemas specific to your application.
- Axis, for example, includes Java exception stack traces.

```
<xs:complexType name="detail">
  <xs:sequence>
    <xs:any namespace="##any"
      processContents="lax"
      minOccurs="0"
      maxOccurs="unbounded" />
    </xs:sequence>
    <xs:anyAttribute
      namespace="##other"
      processContents="lax" />
</xs:complexType>
```

Next Time

- This lecture has examined the basic SOAP message format.
- We have not described the following:
  - The rules for encoding transmitted data
    - Specifically, how do I encode XML for RPC?
    - How does this connect to WSDL?
  - The rules for transmitting messages.
- I also want to give a specific example of extending SOAP to support reliable messaging.