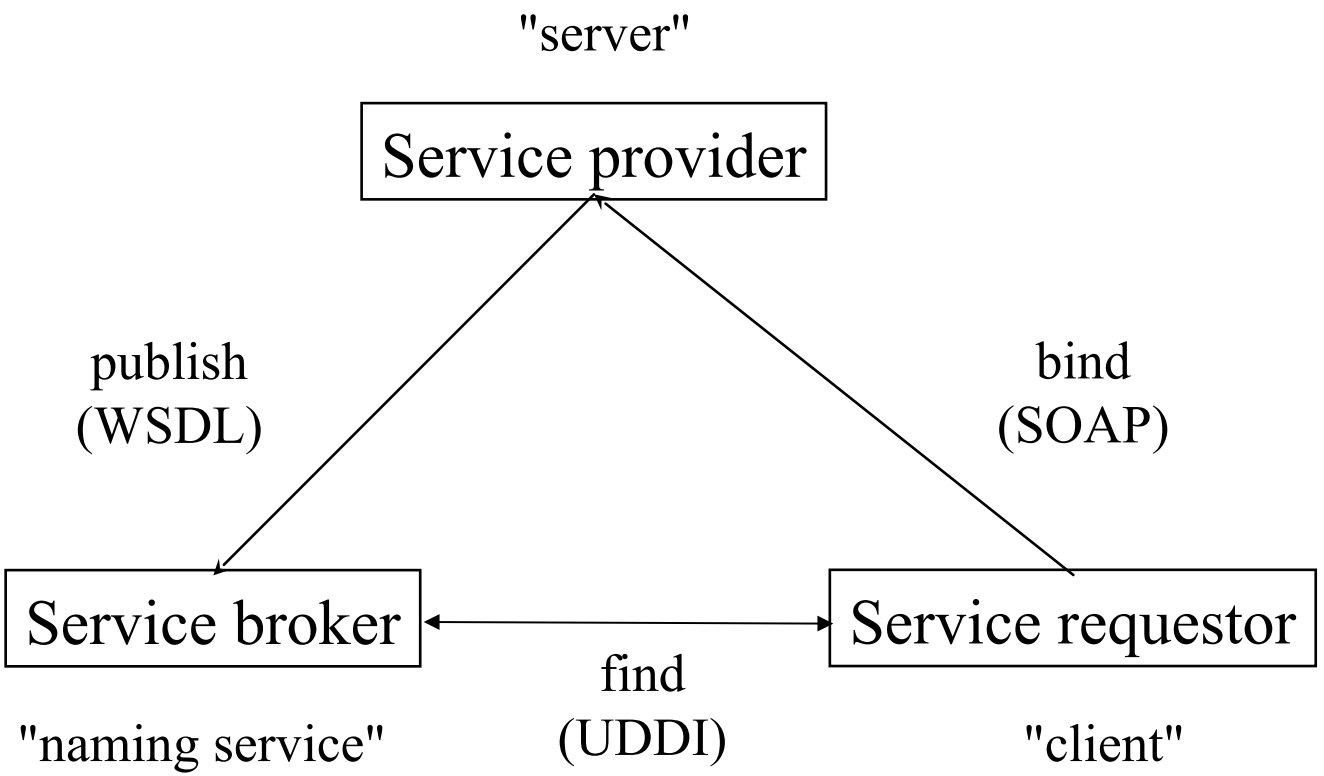
SOAP

Simple Object Access Protocol

Web Evolution

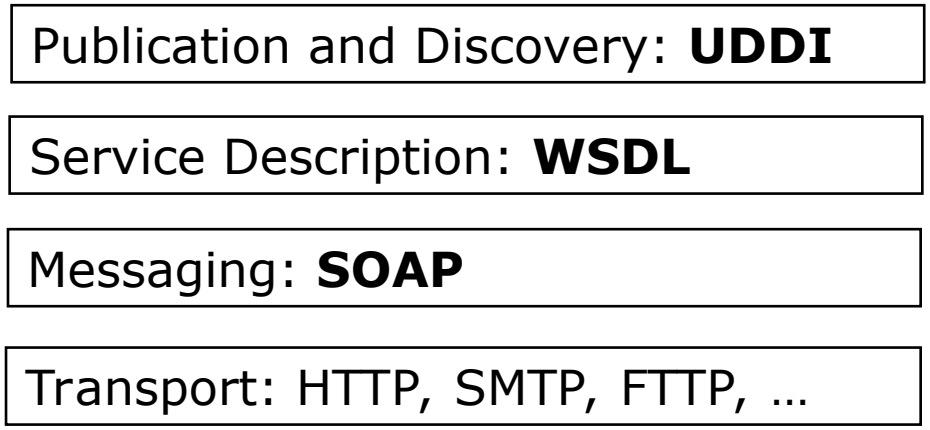
Technology	TCP/IP	HTML	XML
Purpose	Connectivity	Presentation	Programmability
Applications	E-Mail, FTP…	Web Pages	Web Services
Outcome	Create the Web	Browse the Web	Program the Web

Web Service Architecture

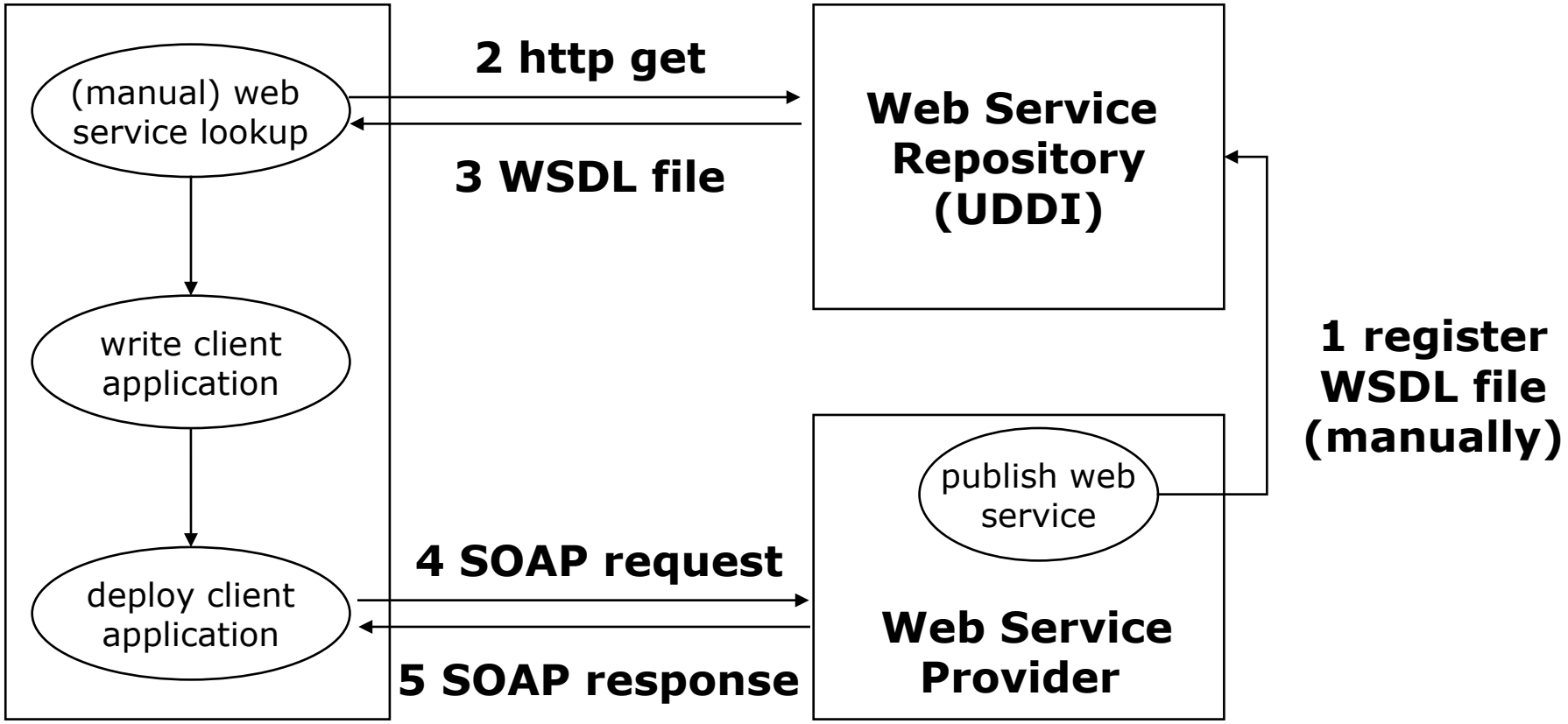


Web Service Stack

A set of standards for implementing web services



Basic Web Service Usage Scenario

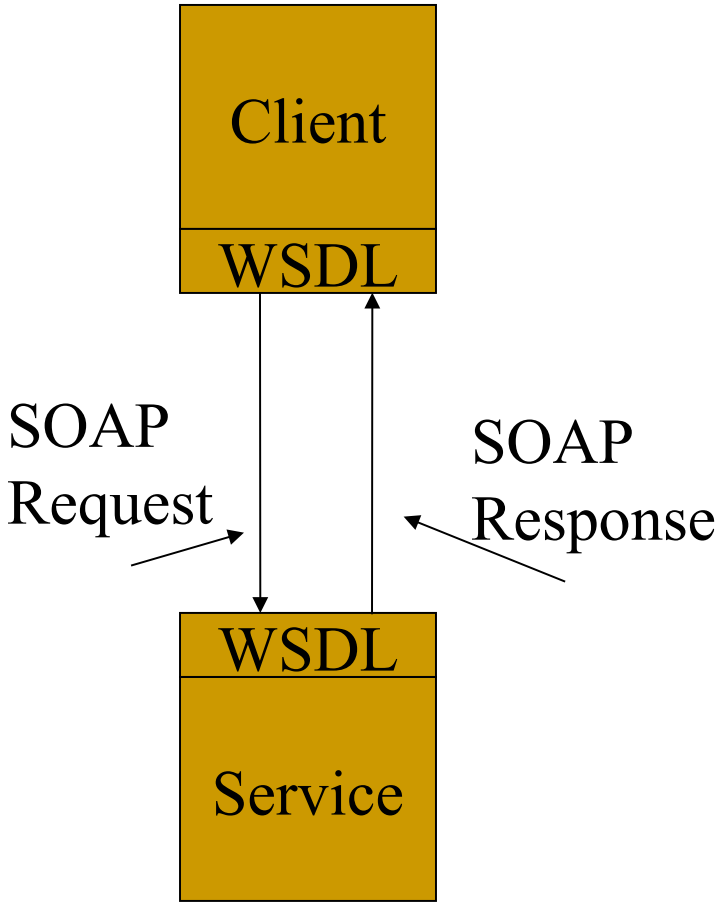


A note on UDDI

- UDDI (launched in 2000) was meant to allow automatic discovery and use of web services
- Very difficult to do in practice
- UDDI is hardly used nowadays
  - Main public UDDI services closed in 2006
  - UDDI dropped from Windows Server after 2010
  - No more work on it
- Information on services now available in simple web pages for human use
  - Including WSDL or OpenAPI specification on how to programmatically connect to them

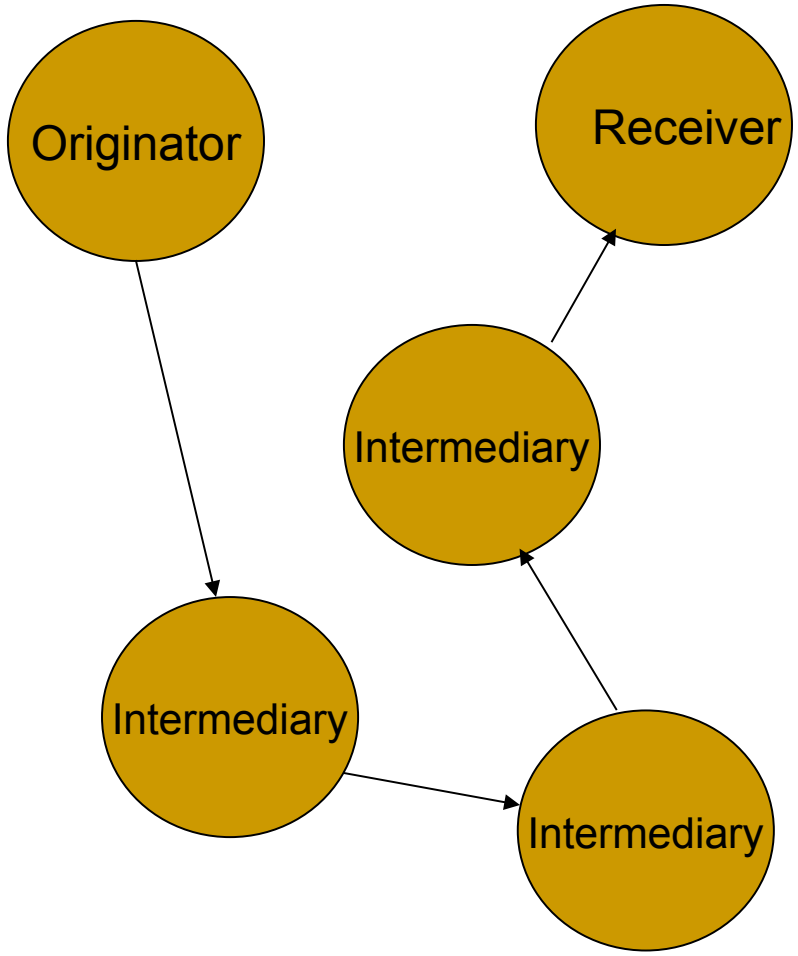
SOAP and Web Services

- In Web Services communications are encoded with SOAP.
  - Transported by HTTP



Beyond Client-Server

- SOAP assumes messages have an *originator*, one or more *ultimate receivers*, and zero or more *intermediaries*.
- The reason is to support distributed message processing.
- That is, we can go beyond client-server messaging.
- Implementing this message routing is out of scope for SOAP.
  - Relies on underlying transport protocol



SOAP in One Slide

- SOAP is just a message format.
  - Must transport with HTTP, TCP, etc.
- SOAP is independent of but can be connected to WSDL.
- SOAP provides rules for processing the message as it passes through multiple steps.
- SOAP payloads
  - SOAP carries arbitrary XML payloads as a body
  - SOAP headers contain any additional information
  - These are encoded using optional conventions

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 and Extensible**
  - Messages may require advanced features like security, reliability, conversational state, etc.
  - Don't design these but do design a place where this sort of advanced information can go
    - Add these capabilities in further specifications: WS-Security, WS-ReliableMessaging, etc.
- **Tell the message originator if something goes wrong (faults)**
- **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

SOAP Messaging

SOAP Basics

- SOAP is often thought of as a protocol extension for doing **Remote Procedure Calls** (RPC) over HTTP.
  - This is how it is often used.
- This is not 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 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 show examples of SOAP message from an Echo service.
  - 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.
- The actual message payload is highlighted.

SOAP Request

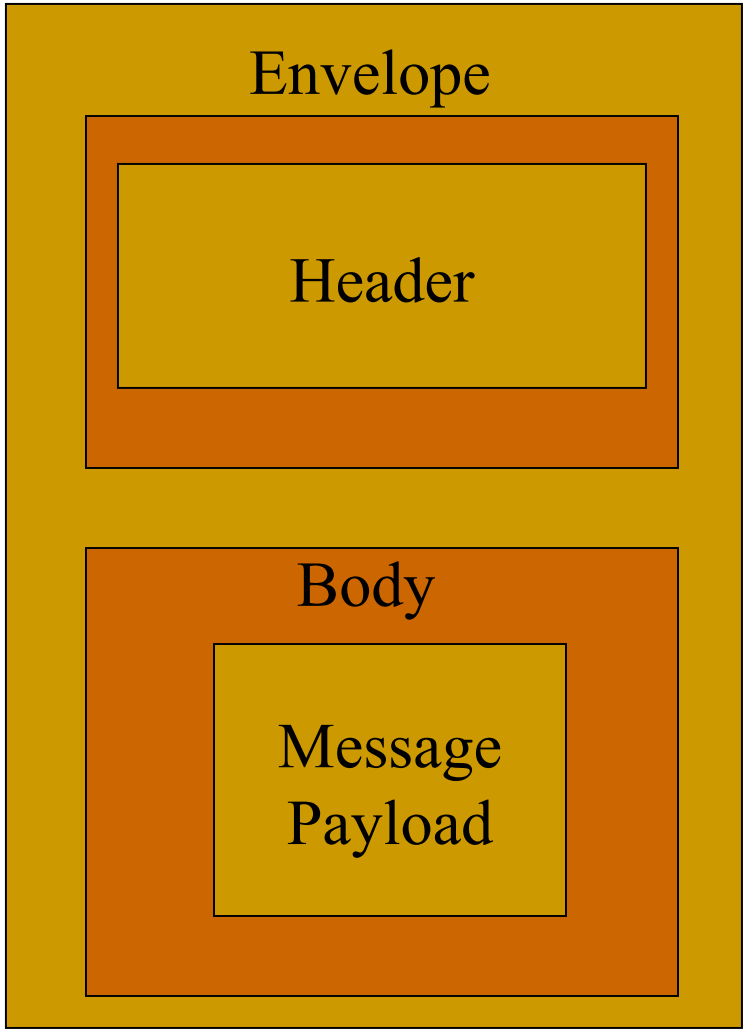
```
<?xml version='1.0' ?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:echo
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"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

SOAP Structure

- SOAP structure is very simple.
  - 0 or 1 header elements
  - 1 body element
  - Envelope 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.
  - Which in turn derives from the SOAP Infoset
- 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
  - lax requires to try to get the schema for the attributes, but no error is generated if it is not found

```
<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>
```

SOAP Envelope

- The envelope is the root container of the SOAP message.
- Things to put in the envelope:
  - 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 needed for XML Schema
  - Encoding rules (optional)
    - Specific rules for deserializing the encoded SOAP data
- Header and body elements.
  - Headers are optional, body is mandatory.
  - Headers come first in the message

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, processing the header element is optional
- Headers may also have a "**relay**" attribute

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 to negotiate particular qualities of service such as reliable message delivery and transactions.
- **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

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
    envelope">
  <env:Header>
    <m:reservation xmlns:m="http://my.example.com/"
    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>
```

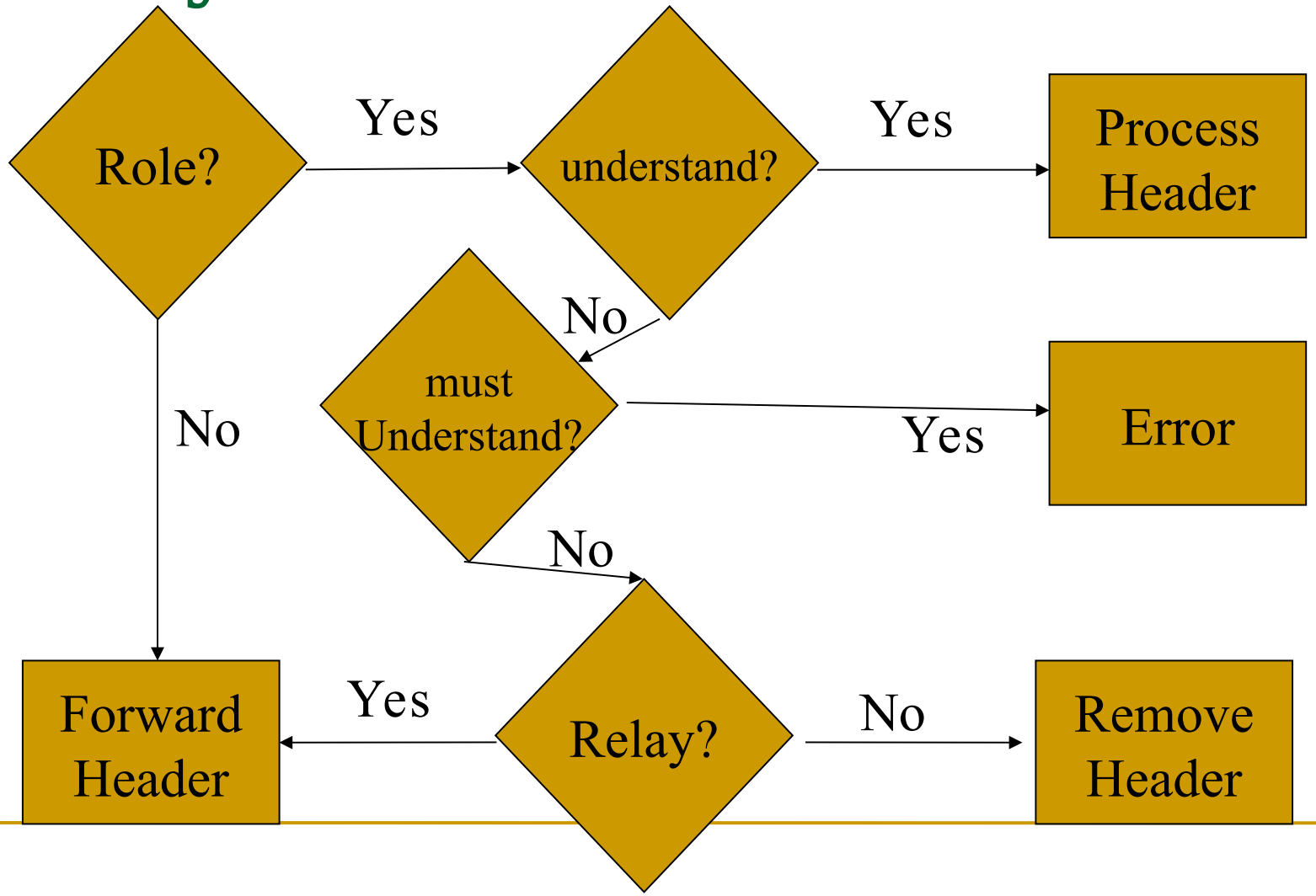
Explanation of Header Example

- In general, we can import tags into the header from name spaces outside of soap.
  - <reservation/>, <reference/>, <dataAndTime/>,<passenger/>
- SOAP doesn't need to worry too much about these.
  - It is the node's job to process these things.
- 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.
  - Imagine SOAP messages being passed through many distinct nodes.
  - 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 provide the required security, session, or reliability requirements, etc.

Roles, Understanding, and Relays



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 Example

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

Example SOAP Body Details

- The <Body> tag includes encoding information.
- This particular style is called RPC.
  - The top-level tag corresponds to the name of the invoked method
  - Internal tags correspond to parameters mapped to XML
- xsi:type is used to specify that the <in0> element takes a string value.

SOAP Fault

- The body may optionally contain a Fault element
- The SOAP Fault element holds errors and status information for a SOAP message.

- A Fault element can appear at most once in a SOAP message.

- The SOAP Fault element has the following sub elements:
  - <faultcode>: A code for identifying the fault
  - <faultstring>: A human readable explanation of the fault
  - <faultactor>: Information about who caused the fault to happen, mainly if it is one of the intermediate nodes
  - <detail>: Holds application specific error information

SOAP Fault codes

- VersionMismatch: Found an invalid namespace for the SOAP Envelope element
- MustUnderstand: An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
- Client: The message was incorrectly formed or contained incorrect information
- Server: There was a problem with the server so the message could not proceed

SOAP Encoding

Intro: Encoding Conventions

SOAP header and body tags can be used to contain arbitrary XML

- Specifically, they can contain an arbitrary sequence of tags
- These tags from other schemas can contain child tags and be quite complex.

And that's all it specifies.

SOAP thus does not impose a content model.

Content models are defined by *convention* and are optional.

Encoding Overview

We typically should provide encoding rules along with the message so that the recipient knows how to process them

- Not needed in Jolie, since typing information is provided by interfaces
- Jolie convention is called RPC-literal

SOAP provides some encoding rule definitions:

- http://schemas.xmlsoap.org/soap/encoding/
- But these rules are not required and must be explicitly included.
- Note this is NOT part of the SOAP message schema.

Terminology:

- Serialization: transforming a datum into an XML instance.
- Deserialization: transforming the XML back to the datum.

Specifying Encoding

Encoding is specified using the encodingStyle attribute.

- This is optional
- There may be no encoding style

This attribute can appear in the envelope, body, or headers

- The value is the standard SOAP encoding rules.

Thus, each part may use different encoding rules.

- If present, the envelope has the default value for the message.
- Headers and body elements may override this within their scope.

```
<soapenv:Body>
  <ns1:echo
    soapenv:encodingStyle="http://
    schemas.xmlsoap.org/soap/enc
    oding/"
     xmlns:ns1="…">
    <!--
    The rest of the payload
    -->
</soapenv:Body>
```

Encoding Simple Values

Our echo service exchanges strings. The actual message is encoded like this:

- **<in0 xsi:type="xsd:string">Hello World</in0>**

xsi:type means that <in0> will take string values.

- And string means explicitly xsd:string, or string from the XML schema itself.

In general, all encoded elements should provide xsi:type elements to help the recipient decode the message.

Simple Type Encoding Examples

Java examples

```
int a=3;
float pi=3.14
String s="Hello";
```

SOAP Encoding

```
<a xsi:type="xsd:int">
        10
    </a>
<pi xsi:type="xsd:float">
        3.14
    </pi>
<s xsi:type="xsd:string">
        Hello
    </s>
```

Explanation of Simple Type Encoding

The XML snippets have two namespaces (would be specified in the SOAP envelope typically).

- xsd: the XML schema. Provides definitions of common simple types like floats, ints, and strings.
- xsi: the XML Schema Instance. Provides the definition of the type element.

Basic rule: if an encoding is used, each element must be given a type and a value.

Defining complex data types

If one needs to define data types not available as simple types in XML Schema he can:

- Define them directly, using XML Schema
- Rely on existing definitions

Example for Encoding Arrays in SOAP 1.1

Java Arrays

- int[3] myArray={23,10,32};

Possible SOAP 1.1 Encoding:

```
<myArray xsi:type="SOAP-ENC:Array"
    SOAP-ENC:arrayType="xsd:int[3]">
        <v1>21</v1>
        <v2>10</v2>
        <v3>32</v3>
</myArray>
```

An Explanation

We started out as before, mapping the Java array name to an element and defining an xsi:type.

But there is no array in the XML schema data definitions.

- XSD doesn't preclude it, but it is a complex type to be defined elsewhere.
- The SOAP encoding schema defines it.

We also made use of the SOAP encoding schema's arrayType attribute to specify the type of array (3 integers).

We then provide the values.