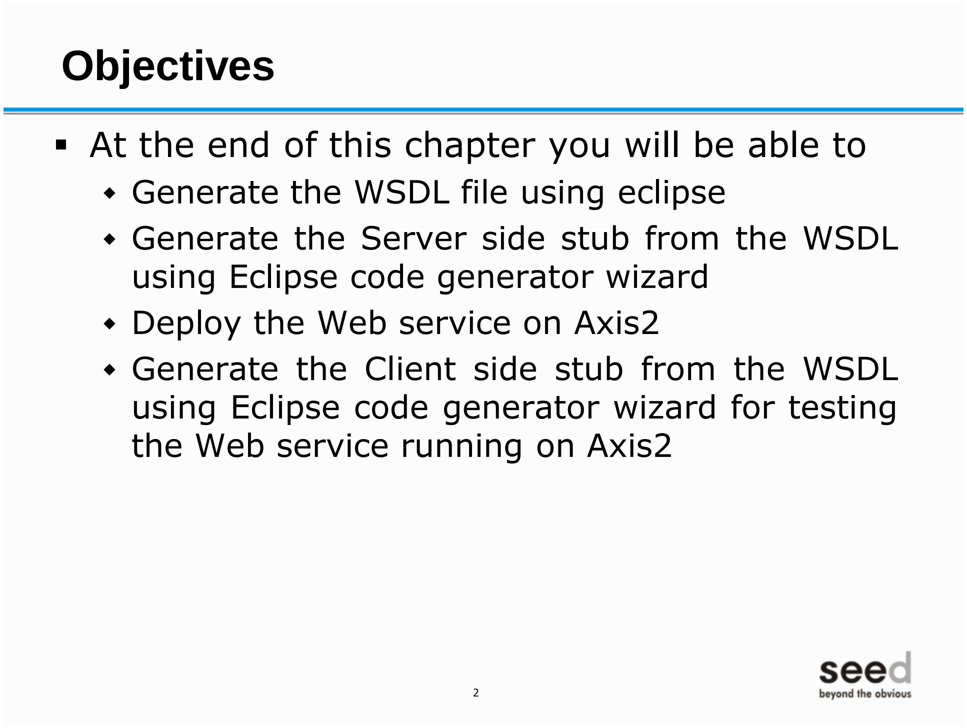
Chapter -

Implementing a web service



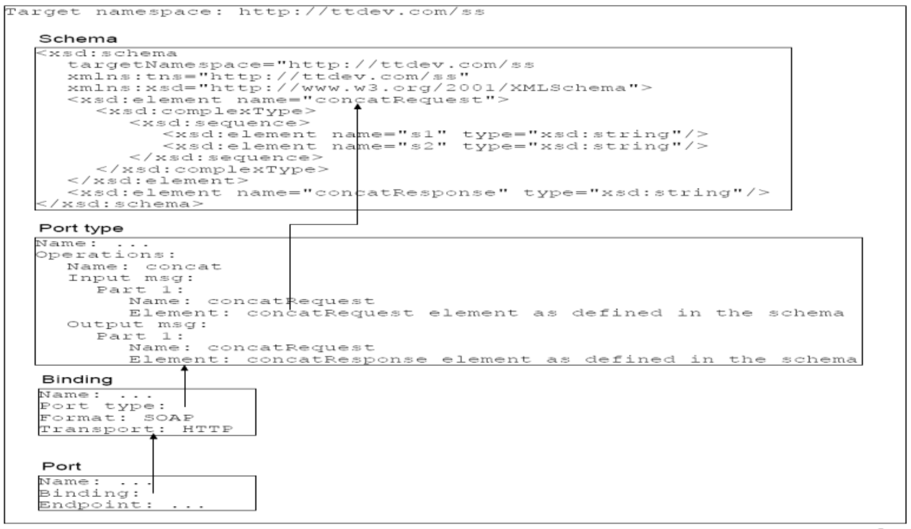
Objectives

- At the end of this chapter you will be able to
  - Generate the WSDL file using eclipse
  - Generate the Server side stub from the WSDL using Eclipse code generator wizard
  - Deploy the Web service on Axis2
  - Generate the Client side stub from the WSDL using Eclipse code generator wizard for testing the Web service running on Axis2

WSDL file for the web service

- Suppose that We'd like to create a web service described in the previous session:

WSDL file for the web service



```
Target namespace: http://ttdev.com/ss
```

Schema

```
<xsd:schema
   targetNamespace="http://ttdev.com/ss
   xmlns:tns="http://ttdev.com/ss"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="s1" type="xsd:string"/>
            <xsd:element name="s2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

Port type

```
Name: ...
Operations:
   Name: concat
   Input msg:
      Part 1:
         Name: concatRequest
         Element: concatRequest element as defined in the schema
   Output msg:
      Part 1:
         Name: concatRequest
         Element: concatResponse element as defined in the schema
```

Binding

```
Name: ...
Port type:
Format: SOAP
Transport: HTTP
```

Port

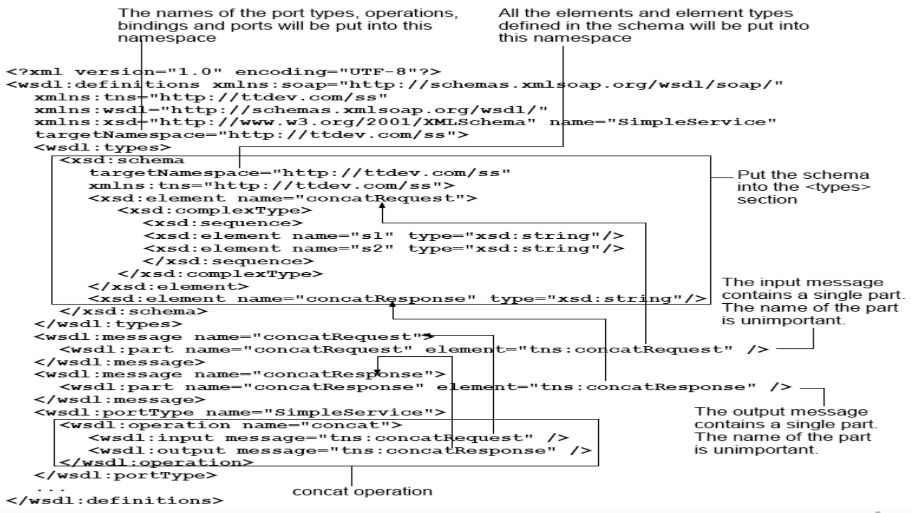
```
Name: ...
Binding:
Endpoint: ...
```

WSDL file for the web service

- To write it using the real WSDL language, it should be:

5

WSDL file for the web service


The names of the port types, operations, bindings and ports will be put into this namespace
All the elements and element types defined in the schema will be put into this namespace
Put the schema into the <types> section
The input message contains a single part. The name of the part is unimportant.
The output message contains a single part. The name of the part is unimportant.
concat operation

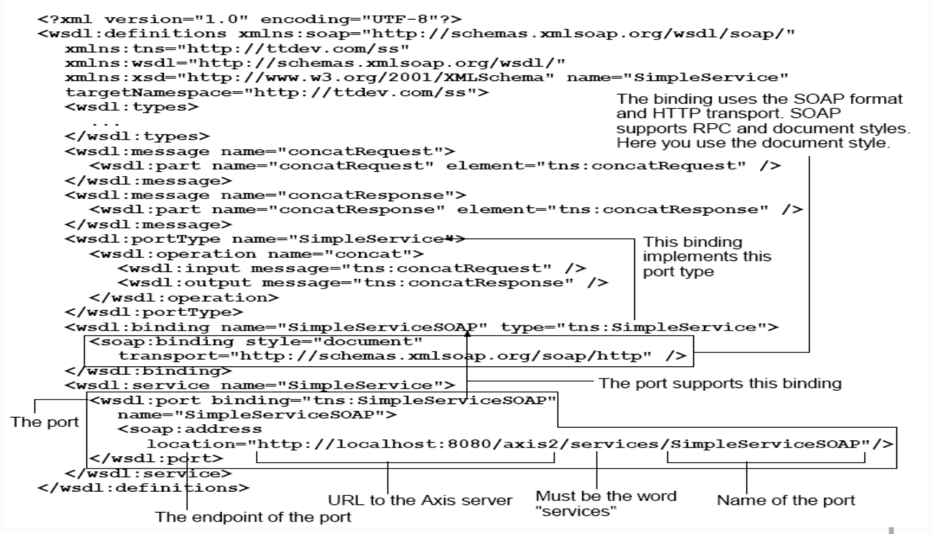
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://ttdev.com/ss"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SimpleService"
   targetNamespace="http://ttdev.com/ss">
   <wsdl:types>
     <xsd:schema
        targetNamespace="http://ttdev.com/ss"
        xmlns:tns="http://ttdev.com/ss">
        <xsd:element name="concatRequest">
           <xsd:complexType>
             <xsd:sequence>
             <xsd:element name="s1" type="xsd:string"/>
             <xsd:element name="s2" type="xsd:string"/>
             </xsd:sequence>
           </xsd:complexType>
        </xsd:element>
        <xsd:element name="concatResponse" type="xsd:string"/>
     </xsd:schema>
   </wsdl:types>
   <wsdl:message name="concatRequest">
     <wsdl:part name="concatRequest" element="tns:concatRequest" />
   </wsdl:message>
   <wsdl:message name="concatResponse">
     <wsdl:part name="concatResponse" element="tns:concatResponse" />
   </wsdl:message>
   <wsdl:portType name="SimpleService">
     <wsdl:operation name="concat">
        <wsdl:input message="tns:concatRequest" />
        <wsdl:output message="tns:concatResponse" />
     </wsdl:operation>
   </wsdl:portType>
   ...
</wsdl:definitions>
```

6

WSDL file for the web service

- Previous presentation defines the schema and the port type. To define the binding and the port:

WSDL file for the web service



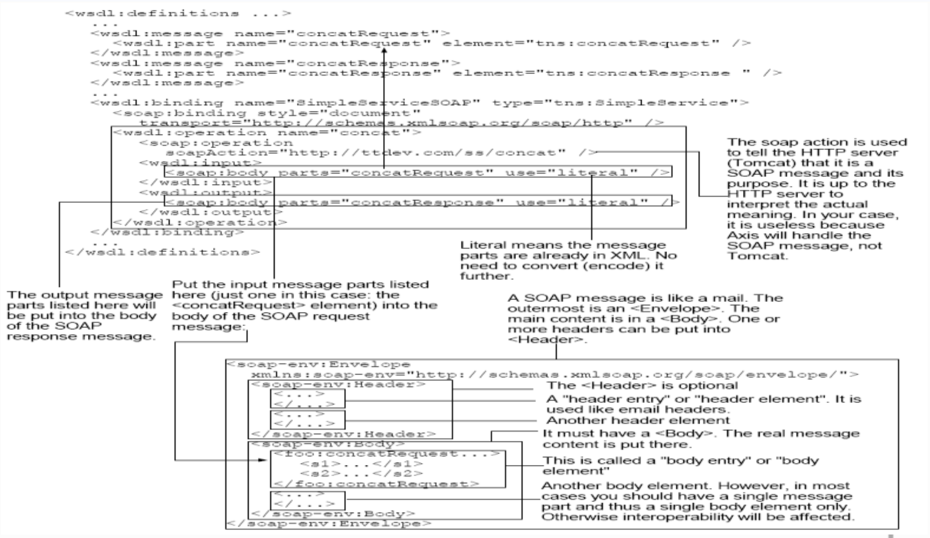
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://ttdev.com/ss"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SimpleService"
  targetNamespace="http://ttdev.com/ss">
  <wsdl:types>
    ...
  </wsdl:types>
  <wsdl:message name="concatRequest">
    <wsdl:part name="concatRequest" element="tns:concatRequest" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="concatResponse" element="tns:concatResponse" />
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
  </wsdl:binding>
  <wsdl:service name="SimpleService">
    <wsdl:port binding="tns:SimpleServiceSOAP"
      name="SimpleServiceSOAP">
      <soap:address
        location="http://localhost:8080/axis2/services/SimpleServiceSOAP"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

The binding uses the SOAP format and HTTP transport. SOAP supports RPC and document styles. Here you use the document style.
This binding implements this port type
The port supports this binding
The port
URL to the Axis server
Must be the word "services"
Name of the port
The endpoint of the port

WSDL file for the web service

- In fact, in a SOAP binding, we need to specify some more details:

WSDL file for the web service



```
<wsdl:definitions ...>
  ...
  <wsdl:message name="concatRequest">
    <wsdl:part name="concatRequest" element="tns:concatRequest" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="concatResponse" element="tns:concatResponse " />
  </wsdl:message>
  ...
  <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="concat">
      <soap:operation
        soapAction="http://ttdev.com/ss/concat" />
      <wsdl:input>
        <soap:body parts="concatRequest" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body parts="concatResponse" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  ...
</wsdl:definitions>
```

The soap action is used to tell the HTTP server (Tomcat) that it is a SOAP message and its purpose. It is up to the HTTP server to interpret the actual meaning. In your case, it is useless because Axis will handle the SOAP message, not Tomcat.

Literal means the message parts are already in XML. No need to convert (encode) it further.

The output message parts listed here will be put into the body of the SOAP response message.

Put the input message parts listed here (just one in this case: the <concatRequest> element) into the body of the SOAP request message:

A SOAP message is like a mail. The outermost is an <Envelope>. The main content is in a <Body>. One or more headers can be put into <Header>.

```
<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    <...>
    </...>
    <...>
    </...>
  </soap-env:Header>
  <soap-env:Body>
    <foo:concatRequest...>
      <s1>...</s1>
      <s2>...</s2>
    </foo:concatRequest>
    <...>
    </...>
  </soap-env:Body>
</soap-env:Envelope>
```

The <Header> is optional

A "header entry" or "header element". It is used like email headers.

Another header element

It must have a <Body>. The real message content is put there.

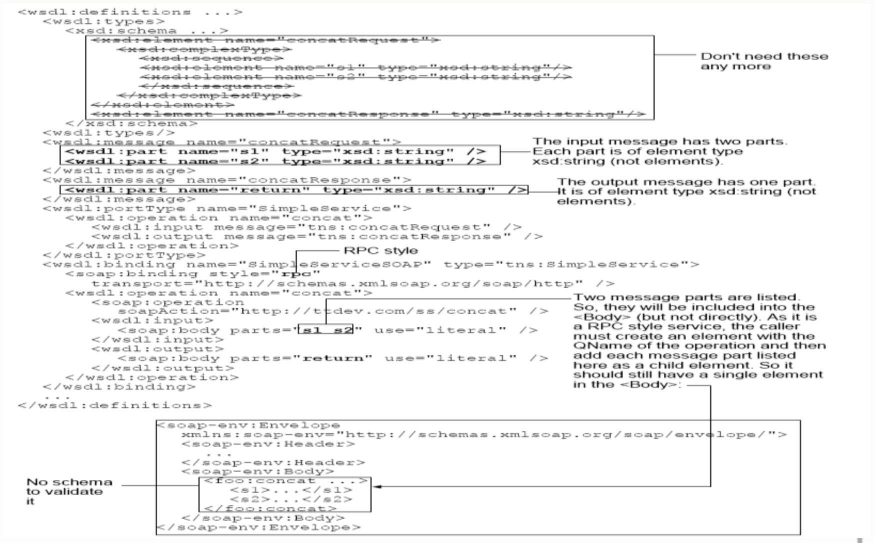
This is called a "body entry" or "body element"

Another body element. However, in most cases you should have a single message part and thus a single body element only. Otherwise interoperability will be affected.

RPC version of the web service

- If the web service was a RPC style service, then the WSDL file would be like:

RPC version of the web service



```
<wsdl:definitions ...>
  <wsdl:types>
    <xsd:schema ...>
      ~~<xsd:element name="concatRequest">~~
        ~~<xsd:complexType>~~
          ~~<xsd:sequence>~~
          ~~<xsd:element name="s1" type="xsd:string"/>~~
          ~~<xsd:element name="s2" type="xsd:string"/>~~
          ~~</xsd:sequence>~~
        ~~</xsd:complexType>~~
      ~~</xsd:element>~~
      ~~<xsd:element name="concatResponse" type="xsd:string"/>~~
    </xsd:schema>
  <wsdl:types/>
  <wsdl:message name="concatRequest">
    <wsdl:part name="s1" type="xsd:string" />
    <wsdl:part name="s2" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="return" type="xsd:string" />
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleServiceSOAP" type="tns:SimpleService">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="concat">
      <soap:operation
        soapAction="http://ttdev.com/ss/concat" />
      <wsdl:input>
        <soap:body parts="s1 s2" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body parts="return" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  ...
</wsdl:definitions>
```

Don't need these any more

The input message has two parts. Each part is of element type xsd:string (not elements).

The output message has one part. It is of element type xsd:string (not elements).

RPC style

Two message parts are listed. So, they will be included into the <Body> (but not directly). As it is a RPC style service, the caller must create an element with the QName of the operation and then add each message part listed here as a child element. So it should still have a single element in the <Body>:

```
<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    ...
  </soap-env:Header>
  <soap-env:Body>
    <foo:concat ...>
      <s1>...</s1>
      <s2>...</s2>
    </foo:concat>
  </soap-env:Body>
</soap-env:Envelope>
```

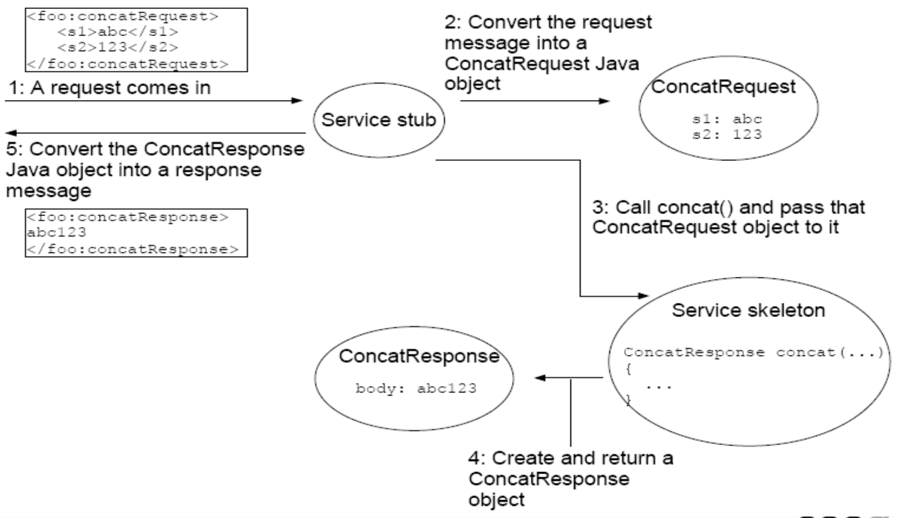
No schema to validate it

Demo

- Demo:- Create a **SimpleService**, a web service providing two string concatenation service.
- Demo Steps:-
  - Create a WSDL file
  - Generate a server side code from WSDL
  - Deploy the service on Apache Axis2
  - Generate the client side stub from WSDL
  - Run the client
- Demo workspace:- .\eclipse-workspace2

seed
beyond the obvious

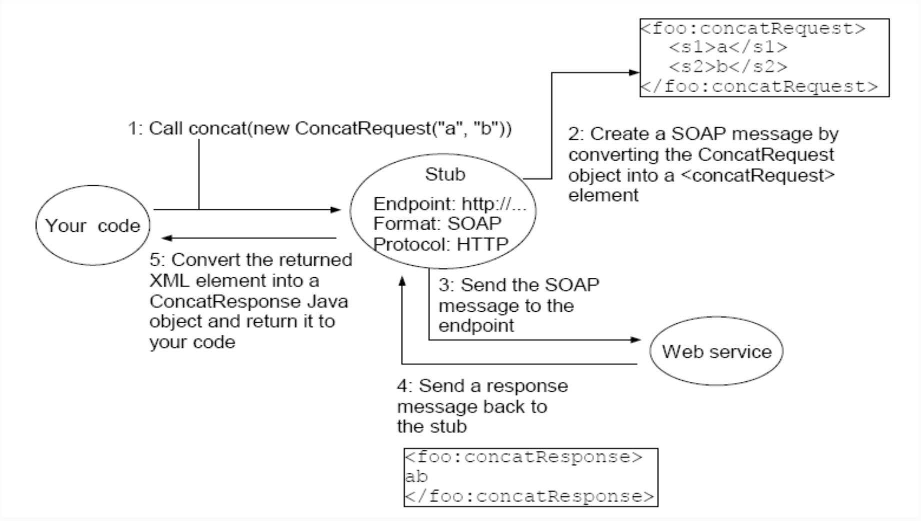
Working of a service stub



<foo:concatRequest>
<s1>abc</s1>
<s2>123</s2>
</foo:concatRequest>
1: A request comes in
2: Convert the request message into a ConcatRequest Java object
Service stub
ConcatRequest
s1: abc
s2: 123
5: Convert the ConcatResponse Java object into a response message
<foo:concatResponse>
abc123
</foo:concatResponse>
3: Call concat() and pass that ConcatRequest object to it
Service skeleton
ConcatResponse concat(...)
{
...
}
ConcatResponse
body: abc123
4: Create and return a ConcatResponse object

seed
beyond the obvious

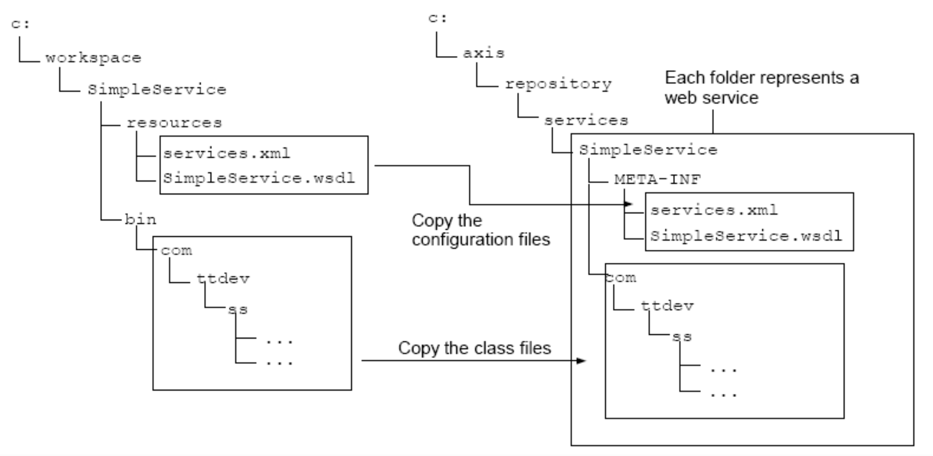
Working of a client



<foo:concatRequest>
<s1>a</s1>
<s2>b</s2>
</foo:concatRequest>
1: Call concat(new ConcatRequest("a", "b"))
2: Create a SOAP message by converting the ConcatRequest object into a <concatRequest> element
Stub
Endpoint: http://...
Format: SOAP
Protocol: HTTP
Your code
5: Convert the returned XML element into a ConcatResponse Java object and return it to your code
3: Send the SOAP message to the endpoint
Web service
4: Send a response message back to the stub
<foo:concatResponse>
ab
</foo:concatResponse>

15

Deployment structure of the web service



16

Quick Recap . . .

- To create a web service, we first create a WSDL file describing its interface.
- This can be done manually or using a tool like Eclipse.
- Then use the Axis Code Generator Wizard on the WSDL file to generate a service stub.
- Then fill in the code in the service skeleton.
- The service stub will convert the XML elements in a request message into Java data/objects, call our skeleton and convert the Java objects returned into XML elements and put them into the response message.

Quick Recap . . .

- The endpoint of the deployed web service is *http://localhost:8080/axis2/services/<name-of-your-service>*.
- To call a web service, run the Axis Code Generator Wizard on the WSDL file to generate a client stub.
- Then, in your code create an instance of the client stub and call its methods as if it were the web service.
- The client stub will convert the Java data/objects into XML elements, create the request message in the right format, send it to the right endpoint using the right transport protocol and convert the XML elements in the response message back into Java data/objects.