Components of WSDL
(Web Service Description Language)

Prof. Vipul Dabhi
Dept. of Information Technology,
D. D. University, Nadiad

What is WSDL?

- Web Service Description Language
- WSDL is a document written in XML
- The document describes a Web service
- Specifies the location of the service and the methods the service exposes

Why WSDL?

- Without WSDL, calling syntax must be determined from documentation that must be provided, or from examining wire messages
- With WSDL, the generation of proxies for Web services is automated in a truly language- and platform-independent way

Where does WSDL fit?

- SOAP is the envelope containing the message
- WSDL describes the service
- UDDI is a listing of web services described by WSDL

Document Structure

- Written in XML
- Two types of sections
  - Abstract and Concrete
- *Abstract* sections define SOAP messages in a platform- and language-independent manner
- Site-specific matters such as serialization are relegated to the *Concrete* sections

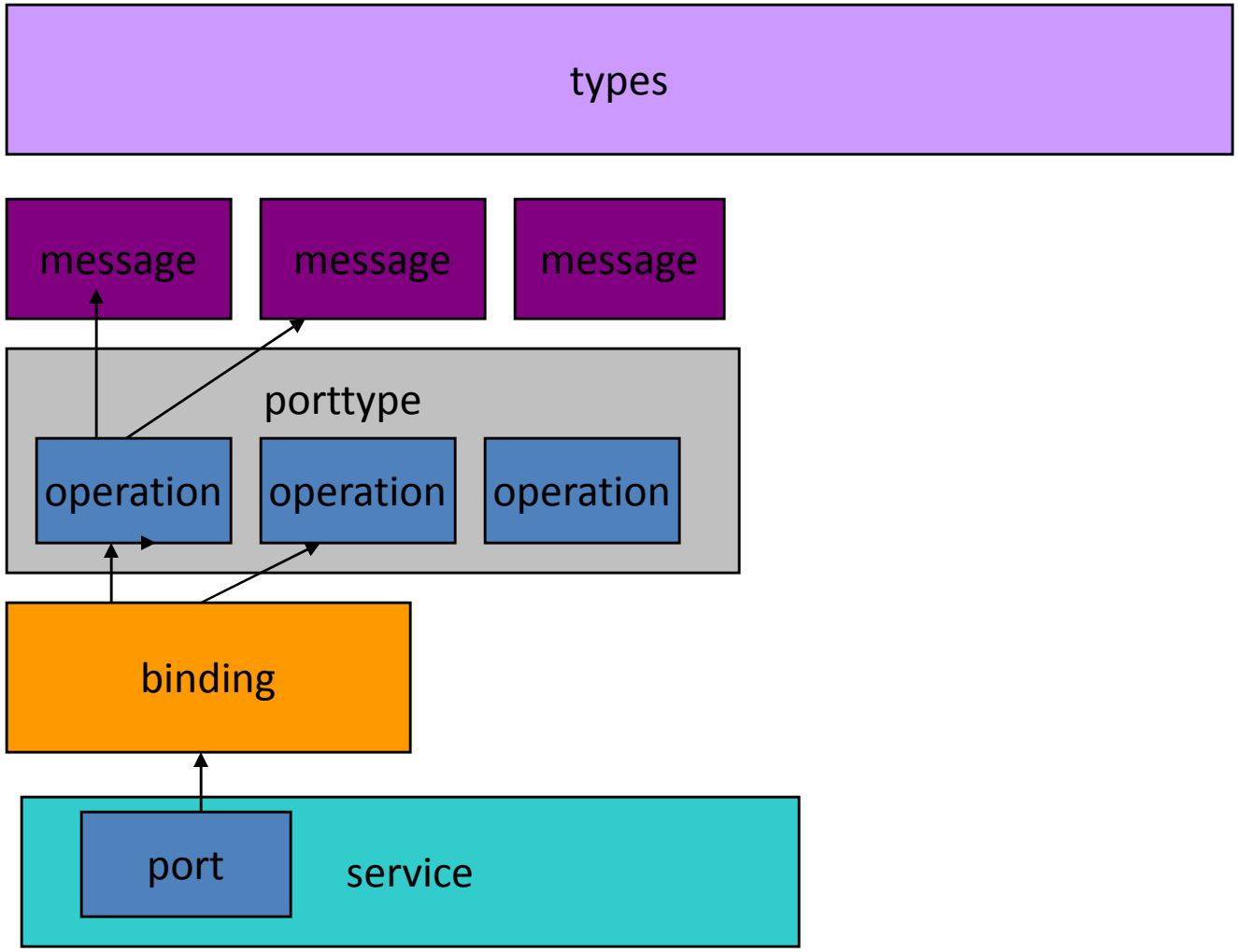
Abstract Definitions

- **Types:** Machine- and language-independent type definitions.
- **Messages:** Contains function parameters (inputs are separate from outputs) or document descriptions.
- **PortTypes:** Refers to message definitions in Messages section that describe function signatures (operation name, input parameters, output parameters).

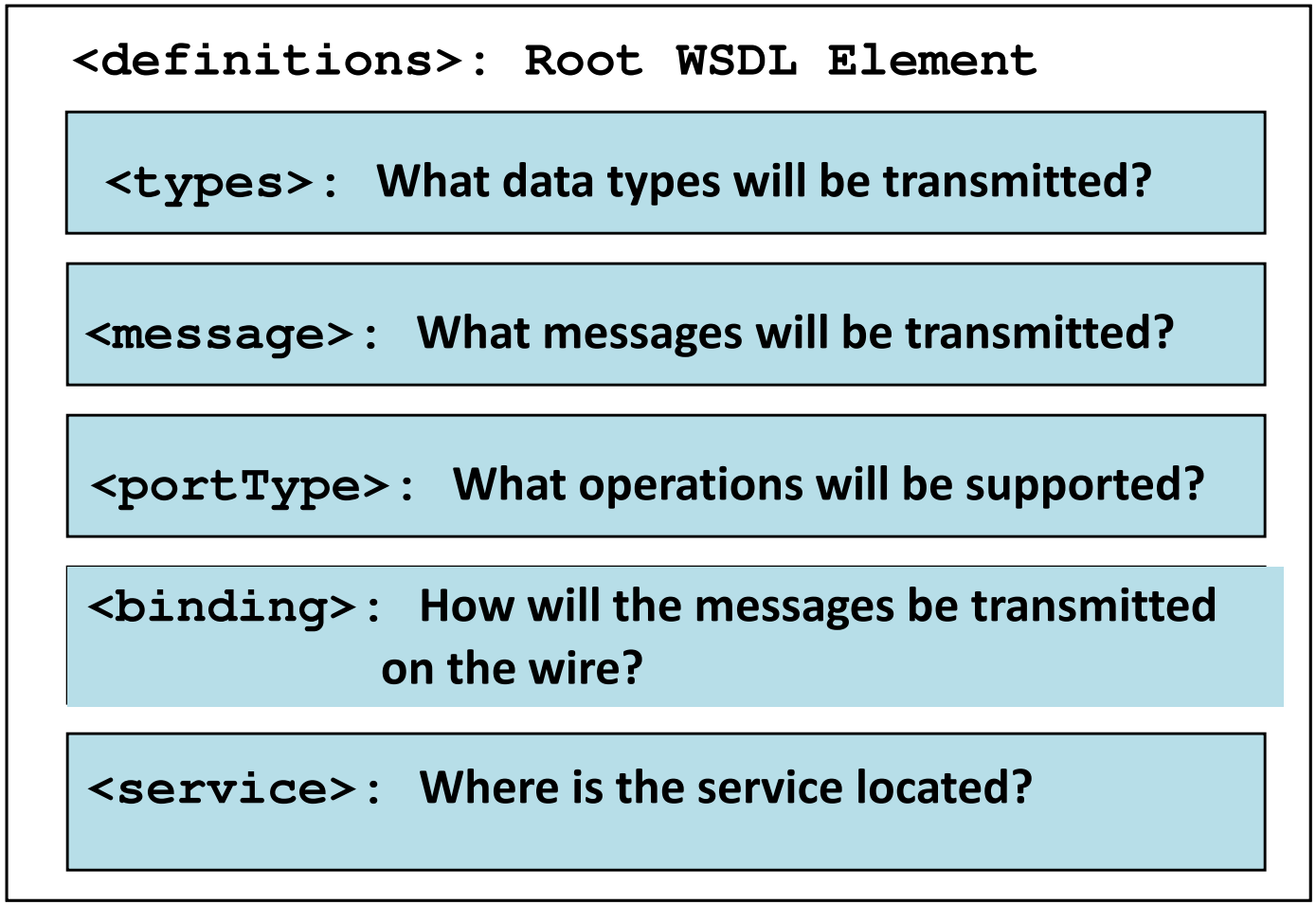
Concrete Descriptions

- **Bindings:** Specifies binding(s) of each operation in the PortTypes section.
- **Services:** Specifies port address(es) of each binding.

WSDL Specification



- A WSDL document can be divided into six major elements



- **`definitions`**
  - Must be the root element
  - Define the name of the service
  - Declare the namespaces used in the document
- **`types`**
  - Describe all the data type used by the Client and Server
  - Can be omitted of only simple data types are used
- **`message`**
  - Define the name of the request/response messages
  - Define also the message part elements
- **`portType`**
  - Define the combination of message elements to form a complete one-way or round-trip operation

- **binding**
  - Provide specific details on how a portType operation will actually be transmitted over the wire
  - SOAP specific information can be defined here. WSDL includes built-in extensions for defining SOAP services
- **service**
  - Define the address for invoking the specified service
- **documentation (less commonly used)**
  - Provide human-readable documentation
  - Similar to making comments in a program
- **import (not all WSDL tools support)**
  - Allow importing other WSDL documents or XML Schemas into a WSDL document
  - Enable a more modular WSDL document

WSDL Structure

- A WSDL document is an XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
   <types>
         <!- define the types here using XML Schema →
   </types>
   <message>
         <!- XML messages the web service uses are defined here →
   </message>
   <portType>
         <!- define the input and output parameters here -→
   </portType>
   <binding>
         <!- define the network protocol here →
   </binding>
   <service>
         <!- location of the service →
   </service>
</definitions>
```

<import> element

```
<definitions
    targetNamespace="urn:3950"
    xmlns= "http://schema.xmlsoap.org/wsdl/"
    xmlns:xsd= "http://www.w3c.org/2001/XMLSchema"
    xmlns:soap= "http://schemas.xmlsoap.org/wsdl/soap/"
    xmlnssoapenc= "http://schemas.xmlsoap.org/soap/emcoding/"
    xmlns:tns= "urn:3950">

    <import namespace= "http://nesc.ac.uk" location= "http://nesc.ac.uk/ez.xsd"/>
```

Acts like C/C++ #include , or Java import.
Incorporates external namespaces

Namespaces

- WSDL uses a number of different namespaces including
- XML Schema Namespaces
  - http://www.w3.org/2000/10/XMLSchema
  - http://www.w3c.org/2001/XML-Schema-instance
- WSDL Namespaces
  - http://schemas.xmlsoap.org/wsdl/soap/
  - http://schemas.xmlsoap.org/wsdl/
- SOAP Namespaces
  - http://schemas.xmlsoap.org/soap/encoding
  - http://schemas.xmlsoap.org/soap/envelope

definitions

targetNamespace is the logical namespace for information about this service. WSDL documents can import other WSDL documents, and setting targetNamespace to a unique value ensures that the namespaces do not clash

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
 targetNamespace=
     "http://localhost:8080/axis/services/NameAndAge"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
      :
      :
>
```

Default namespace. All the WSDL elements, such as <definitions>, <types> and <message> reside in this namespace.

Define the namespaces that will be used in the later part of the document

```
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl=
    "http://localhost:8080/axis/services/NameAndAge"
xmlns:intf=
    "http://localhost:8080/axis/services/NameAndAge"
xmlns:soapenc=
    "http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="enpklun:polyu.edu.hk:soap"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap=
    "http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Types Section

- The *type* element defines the data types that are used by the web service.
- <xsd:complexType name="PERSON">
  <xsd:sequence>
  <xsd:element name="firstName“ type="xsd:string"/>
  <xsd:element name="lastName" type="xsd:string"/>
  <xsd:element name="ageInYears" type="xsd:int"/>
  </xsd:sequence>
  </xsd:complexType>

types – give details of complex data type

The Name of item, its namespace is defined by targetNameSpace

Default namespace, apply to unspecified tags, e.g. schema, sequence, complexType, element

```
<wsdl:types>
 <schema targetNamespace="enpklun:polyu.edu.hk:soap"
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="Record">
   <sequence>
    <element name="age" type="xsd:int" />
    <element name="name" nullable="true"
                                  type="xsd:string" />
   </sequence>
  </complexType>
 </schema>
</wsdl:types>
```

can be a null string

Two parameters of Record to be sent. The element names are derived from the get/set functions of the JavaBean

The <types>

- The types element contains XML Schemas defining the datatypes that are to be passed to and from the web service

```
<types>
 <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
  <element name="TradePriceRequest">
   <complexType>
     <all><element name="tickerSymbol" type="string"/></all>
   </complexType>
  </element>
  <element name="TradePrice">
    <complexType>
     <all><element name="price" type="float"/></all>
    </complexType>
  </element>
 </schema>
</types>
```

Messages Section

- A *message* element defines parameters
- The name of an output message element ends in "Response" by convention
- <message name="Simple.foo">
  <part name="arg" type="xsd:int"/>
  </message>

  <message name="Simple.fooResponse">
  <part name="result" type="xsd:int"/>
  </message>

More on Messages

- Messages consist of one or more logical parts
- Each part is associated with a type

```
<definitions .... >
 <message name="nmtoken">
        * <part name="nmtoken" element="qname"? type="qname"?/> *
 </message>
 </definitions>
```

More on Messages

- Multiple part elements are used if the message has multiple logical units
- Abstract vs. Concrete messages
  - Message definitions are abstract
  - Message binding describes how the abstract content is mapped to a concrete format
  - Bindings may provide very limited information is they are close

The <message>

- The <**message**> element is used to define the messages that will be exchanged between the client and the service
- These message elements contain <**part**> elements, which will be using types defined in the types element

```
<message name="GetLastTradePriceInput">
   <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
   <part name="body" element="xsd1:TradePrice"/>
</message>
```

- All the parts are namespace qualified

- The namespace of tns1 as defined in “definition” is enpklun:polyu.edu.hk:soap
- The same as the targetNameSpace in “types”
- Hence we are talking about the "Record” described in “types”

```
<wsdl:message name="showRecordRequest">
  <wsdl:part name="in0" type="tns1:Record" />
</wsdl:message>
<wsdl:message name="showRecordResponse">
  <wsdl:part name="showRecordReturn"
                                  type="tns1:Record" />
</wsdl:message>
```

The name of the parameter used in these two messages. Only one in each message

PortTypes Section

- Defines a web service, the operations that can be performed, and the messages that are involved.
- <portType name="SimplePortType">
  <operation name="foo" parameterOrder="arg" >
  <input message="wsdlns:Simple.foo"/>
  <outputmessage="wsdlns:Simple.fooResponse"/>
  </operation>
  </portType>

The <portType>

- The types and messages have been defined, but they have not been defined in terms of where they fit in the functionality of the web service

- This is done within <**portType>** and <**operation>** elements

```
<portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
   </operation>
</portType>
```

- A portType is analogous to a class
- An operation is analogous to a method in that class

portType

- Define how the messages are transmitted for the method: showRecord

```
<wsdl:portType name="RecordService">
  <wsdl:operation name="showRecord"
                                  parameterOrder="in0">
    <wsdl:input message="impl:showRecordRequest"
                            name="showRecordRequest" />
    <wsdl:output message="impl:showRecordResponse"
                            name="showRecordResponse" />
  </wsdl:operation>
</wsdl:portType>
```

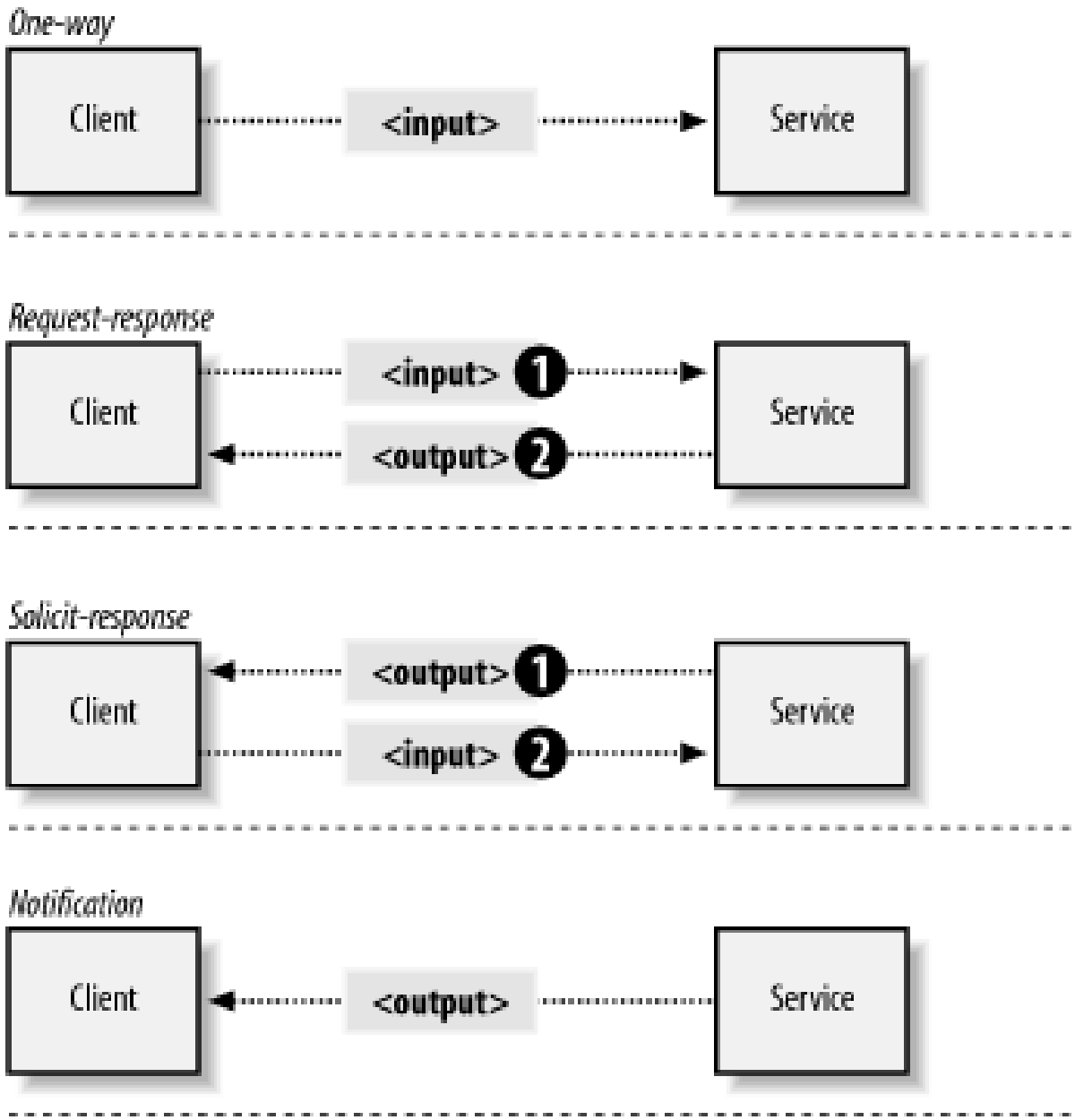
The sequence of the input/output message is matter. The example above means that the input message should go first and followed by the output message

Types of <operation>

- There are four distinct types of operation
- Synchronous
  - **Request-response** - The service receives a message and sends a reply
  - **Solicit-response** - The service sends a message and receives a reply message
- Asynchronous
  - **One-way** - The service receives a message
  - **Notification** - The service sends a message
- All of these can be defined in WSDL

Four operation patterns supported by WSDL 1.1

1. **One-way**
2. **Request-response**
3. **Solicit-response**
4. **Notification**



```
<wsdl:operation name="showRecord"
                                    parameterOrder="in0">
```

- A message can have more than one *"parts"*
  - E.g. if showRecord() requires three input parameters, then the input message for calling the service will have three parts
- For message that has more than one *"parts"*, need to indicate their order, e.g. which part is the first parameter and which part is the second
- Assume the input message of showRecord() has three *"parts"* – in0, in1 and in2, and in0 is the first, in1 is the second and in2 is the third, then

```
<wsdl:operation name="showRecord"
                              parameterOrder="in0 in1 in2">
```

The <binding> element

- This element is used to define the mechanism that the client will actually use to interact with the web service
- There are three possibilities
  1. SOAP
  2. HTTP
  3. MIME
- The most common choice is currently SOAP
- The binding element defines the protocol specific information for the portTypes previously defined

binding

- The binding element provides specific details on how a portType operation will actually be transmitted over the wire
- A single portType can have multiple bindings using different transports e.g. HTTP or SMTP
- Contain the following parts:
  - binding type
  - soap operation
    - function name to be called
    - details about the input parameters
    - details about the return parameters

Talking about the showRecord() of RecordService

```
<wsdl:binding name="NameAndAgeSoapBinding"
                                  type="impl:RecordService">
 <wsdlsoap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="showRecord">
   <wsdlsoap:operation soapAction="" />
   <wsdl:input name="showRecordRequest">
       :
   </wsdl:input>
   <wsdl:output name="showRecordResponse">
       :
   </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
```

using HTTP

Referring to the same operation as in the portType, since same namespace

The binding tag

```
<binding name="ez3950SOAPBinding" type="tns:ez3950PortTypes">
```

The `<binding>` tag indicates that we will map a `<Port Type>` to a protocol

```
<soap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http/">
```

Indicates we will be using the SOAP binding extensions to map the operations.
The alternative to "rpc" is "document".

(to use GET/POST use `<http:binding…>`
to use MIME use `<mime:binding…..>` *)*

Bindings Section

- The *binding* element defines the message format and protocol details for each port.
- <operation name="foo">
  <soap:operation soapAction="http://tempuri.org/action/Simple.foo"/>
  <input>
  <soap:body use="encoded" namespace="http://tempuri.org/message/"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
  <output>
  <soap:body use="encoded“ namespace="http://tempuri.org/message/“
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
  </operation>

The Port Element

- Each <port> element associates a location with a <binding> in a one-to-one fashion
- <port name="fooSamplePort" binding="fooSampleBinding">
  <soap:address location="http://carlos:8080/fooService/foo.asp"/>
  </port>

Services Section

- A collection of related endpoints, where an endpoint is defined as a combination of a binding and an address
- <service name="FOOSAMPLEService">
  <port name="SimplePort“
  binding="wsdlns:SimpleBinding">
  <soap:address
  location="http://carlos:8080/FooSample/
  FooSample.asp"/>
  </port>
  </service>

<service>

- The final component of a WSDL file is the <**service**> element
- The <service> element defines <**port**> elements that specify where requests should be sent

```
<service name="StockQuoteService">
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
        <soap:address location="http://example.com/stockquote"/>
    </port>
</service>
```

- The <soap:address> subelement identifies the URL of the service

- The precise content of <port> elements will be dependent upon the mechanism, i.e. SOAP, HTTP or MIME