UNIVERSITY of York
Europe Campus
CITY College

Introduction to WSDL

CCS3341 Cloud Computing

Dr S. Veloudis

Basics

- Stands for Web Services Description Language
- XML serialisation
- Describes web services

- The location(s) of a service
- The methods of a service

Core elements:

- `<types>`
  Defines the (xsd) data types used by a web service
- `<message>`
  Defines the data elements for each operation
- `<portType>`
  Describes the operations offered and the messages involved
- `<binding>`
  Defines the protocol and data format for each port type

```
<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>
```

Elements

portType (interface in WSDL 2.0)

Defines a named set of operations that the Web service exposes

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

- Operations are specified in terms of <operation>
- An operation specifies any input messages (<input> element) that it receives and/or any output messages that it emits (<output> element)

message

Defines the parts of each message and the data type of each part

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
```

- Each part is specified by the <part> element; a part is analogous to a method parameter
- The data type of each part is:
  - An **xsd** built-in types
  - A custom type defined in the <part> element

Message Exchange Patterns (MEPs)

One-way operation

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

Notification

As above but the input message is replaced by an output message

Request – response operation

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Solicit response operation

As above but the input and output message reversed

Elements

types

Defines custom types to be used as message part types

Uses XML Schema

```
<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>
```

Note:
Built-in xsd types:
`string`
`boolean`
`decimal`
`float`
`double`
`duration`
`dateTime`
`time`
`date`
`YearMonth`
…

Elements

binding

The purpose of the binding element is to bind the interface of a service with a concrete messaging communication technology

- A binding element is always associated with a `portType` (`interface`) element
- The binding element includes an operation element for each operation in the corresponding `portType`
- Each such operation element determines the binding of the `portType` operation to which it corresponds
- A single `portType` operation may have more than one binding

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

Note:
1. `name` attribute determines the name of the binding
2. `type` attribute determines the `portType` being bound

Binding Attributes

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

Two attributes:

1. `style` attribute determines whether the SOAP messages are formatted as `document` or `rpc`
2. `transport` attribute determines the protocol over which SOAP is to be used (here over HTTP)

Note: Other protocols over which SOAP can be used include HTTPS, SMTP, FTP

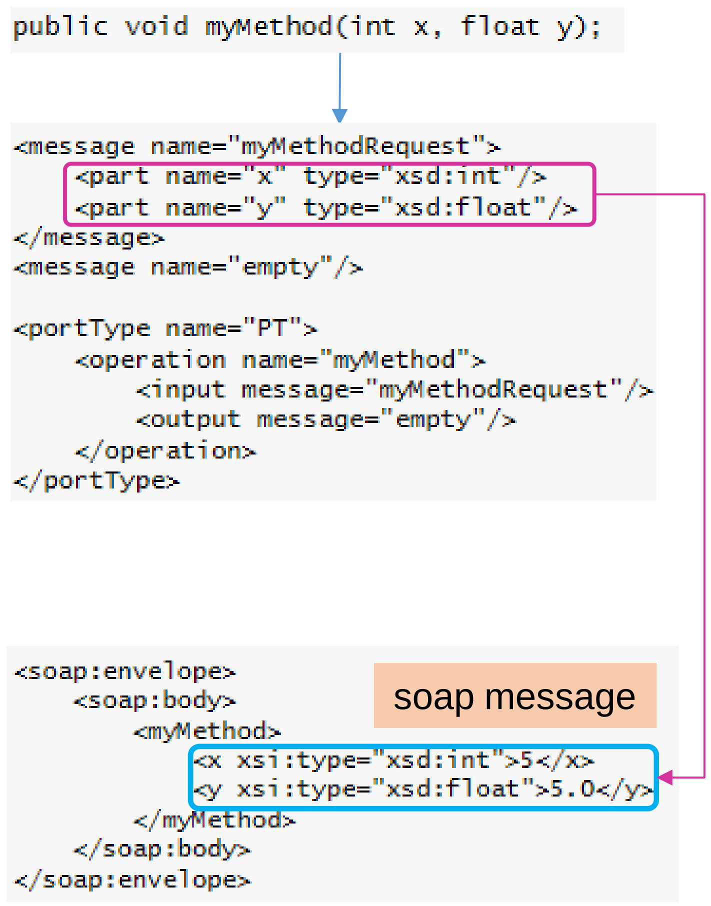
Indicates the intent of the SOAP HTTP request; its value is a URI identifying the endpoint where the resource invoked to serve the operation resides

- Input and output elements mirror the input and output elements of the corresponding operation defined under `portType`
- They contain protocol details that establish how the messages (referenced by the corresponding abstract operation) are going to be processed and interpreted

The `use` attribute can be set to `encoded` or `literal`; together with the `style` attribute they define how the message parts appear inside the SOAP body element

Binding Attributes

`style = rpc and use = encoded`



```
public void myMethod(int x, float y);
```

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>
```

soap message

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Indicates that the SOAP body will contain an XML representation of a method call and that the message parts represent the parameters to this method

Pros:

- Straightforward WSDL
- The operation name appears in the message (makes dispatching this message to the implementation of the operation easy)

Cons:

- The type encoding info in the message is in most cases just overhead which only degrades input performance
- Difficult to validate the message (with an XML validator) since the `soap:body` element contains a method call whose structure is not defined in any XML Schema
- Limited to built-in XSD types

Binding Attributes

```
style = rpc and use = literal
```

Indicates that the SOAP body will contain an XML representation of a method call and that the message parts represent the parameters to this method

```
public void myMethod(int x, float y);
```

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>
```

soap message

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Pros:

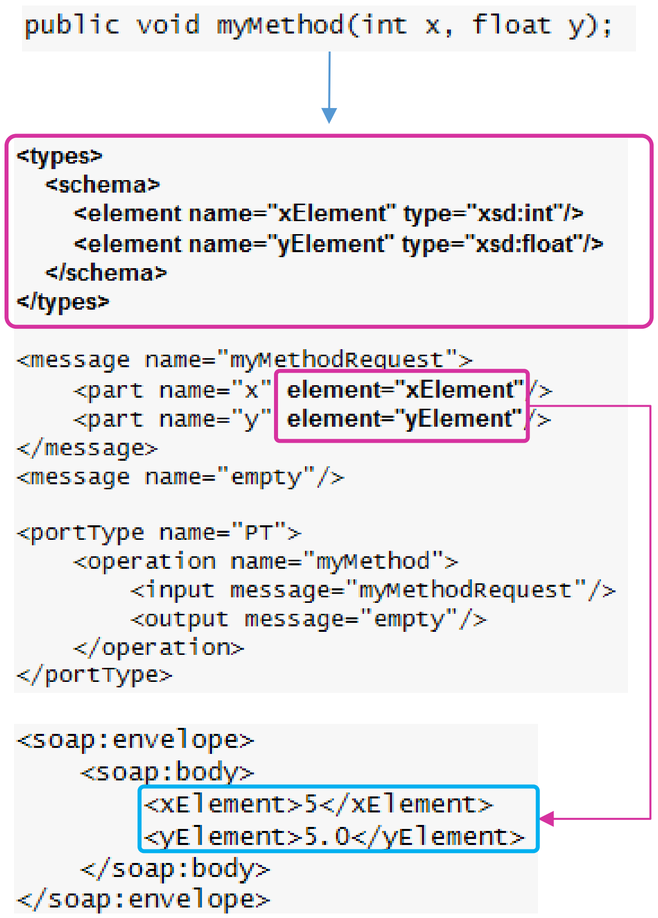
- Straightforward WSDL
- The operation name appears in the message (makes dispatching this message to the implementation of the operation easy)

Cons:

- Difficult to validate the message (with an XML validator) since the `soap:body` element contains a method call whose structure is not defined in any XML Schema
- Limited to built-in XSD types

Binding Attributes

```
style = document and
use = literal
```



```
public void myMethod(int x, float y);
```

```
<types>
  <schema>
    <element name="xElement" type="xsd:int"/>
    <element name="yElement" type="xsd:float"/>
  </schema>
</types>
```

```
<message name="myMethodRequest">
    <part name="x" element="xElement"/>
    <part name="y" element="yElement"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>
```

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

Indicates that the SOAP body will contain an XML document, and that the message parts specify the XML elements that will be placed there

Pros:
- Easy to validate message (by an XML validator) as everything in `soap:body` structure is defined in a schema

Cons:
- More complicated WSDL (for the developer)

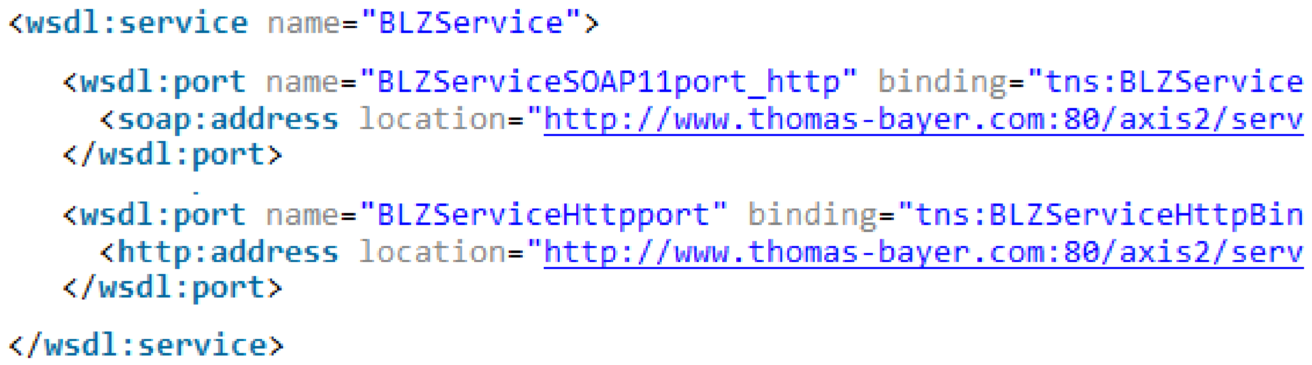
Elements

service

Defines a collection of ports, or endpoints, that expose a particular binding

A single service may be exposed through multiple endpoints

- Each port describes a way to access the service through a particular binding
- This binding (specified through the `binding` attribute) is a binding that has already been defined in the WSDL document



```
<wsdl:service name="BLZService">

   <wsdl:port name="BLZServiceSOAP11port_http" binding="tns:BLZService
     <soap:address location="http://www.thomas-bayer.com:80/axis2/serv
   </wsdl:port>

   <wsdl:port name="BLZServiceHttpport" binding="tns:BLZServiceHttpBin
     <http:address location="http://www.thomas-bayer.com:80/axis2/serv
   </wsdl:port>

</wsdl:service>
```

The `address` element in each port has one attribute, namely `location`, pointing to an endpoint address of the service

UNIVERSITY of York
Europe Campus
CITY College

Elements

definitions

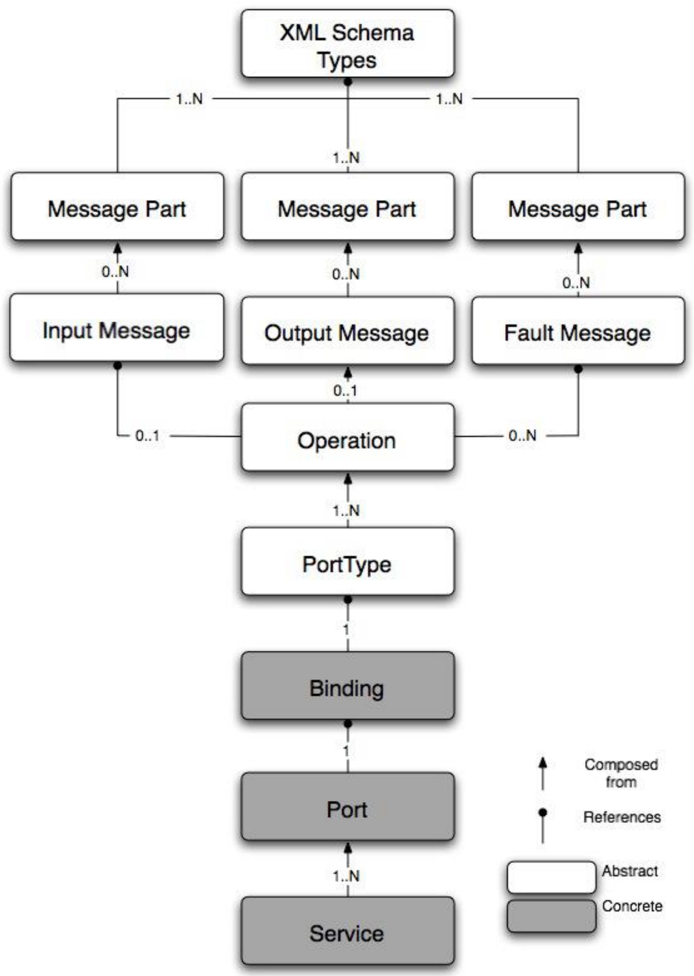
Root element of any WSDL document; defined namespace endpoints

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

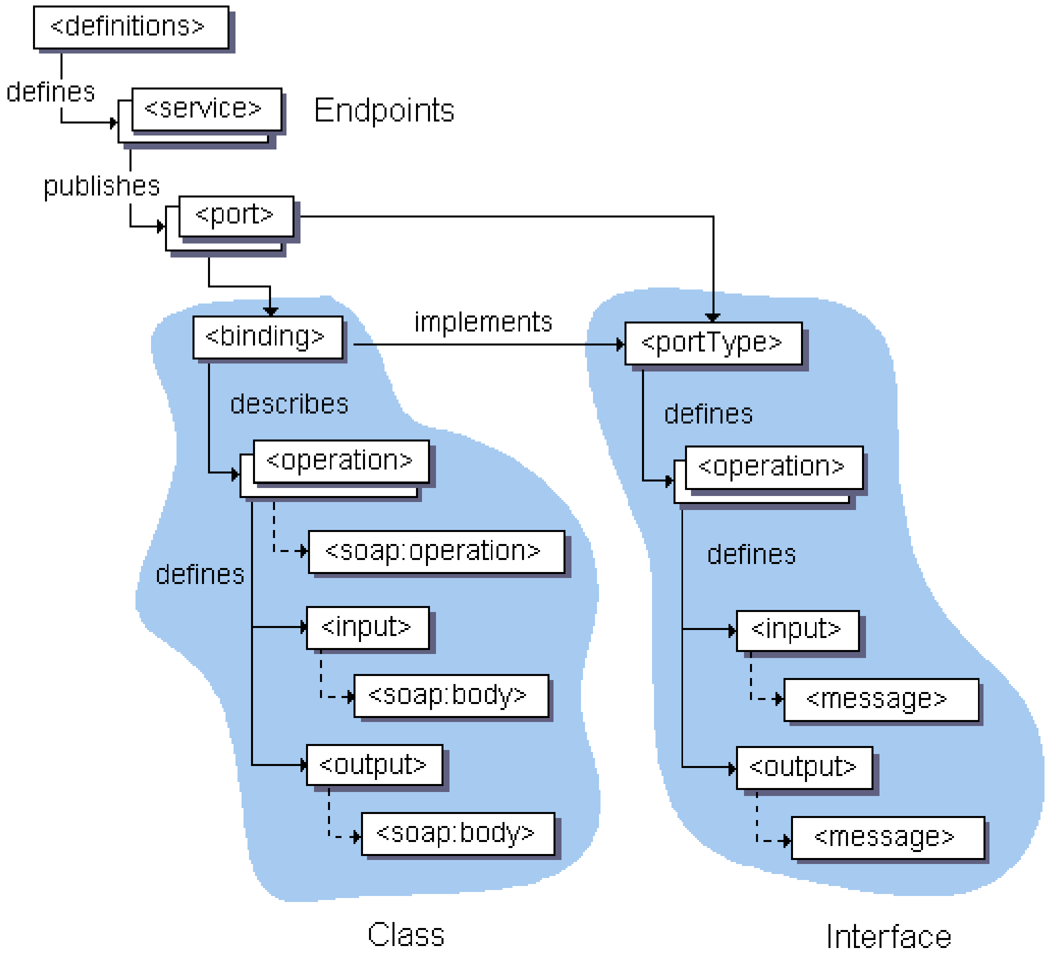
Abbreviations

- `targetNamespace`
  The namespace of the service; all elements defined in the WSDL document are put into this namespace
- `xmlns`
  Default namespace

Recap



XML Schema Types
1..N
1..N
1..N
Message Part
Message Part
Message Part
0..N
0..N
0..N
Input Message
Output Message
Fault Message
0..1
0..1
0..N
Operation
1..N
PortType
1
Binding
1
Port
1..N
Service
Composed from
References
Abstract
Concrete



<definitions>
defines
<service>
Endpoints
publishes
<port>
<binding>
implements
<portType>
describes
defines
<operation>
<operation>
<soap:operation>
defines
defines
<input>
<input>
<soap:body>
<message>
<output>
<output>
<soap:body>
<message>
Class
Interface