SOAP web-services

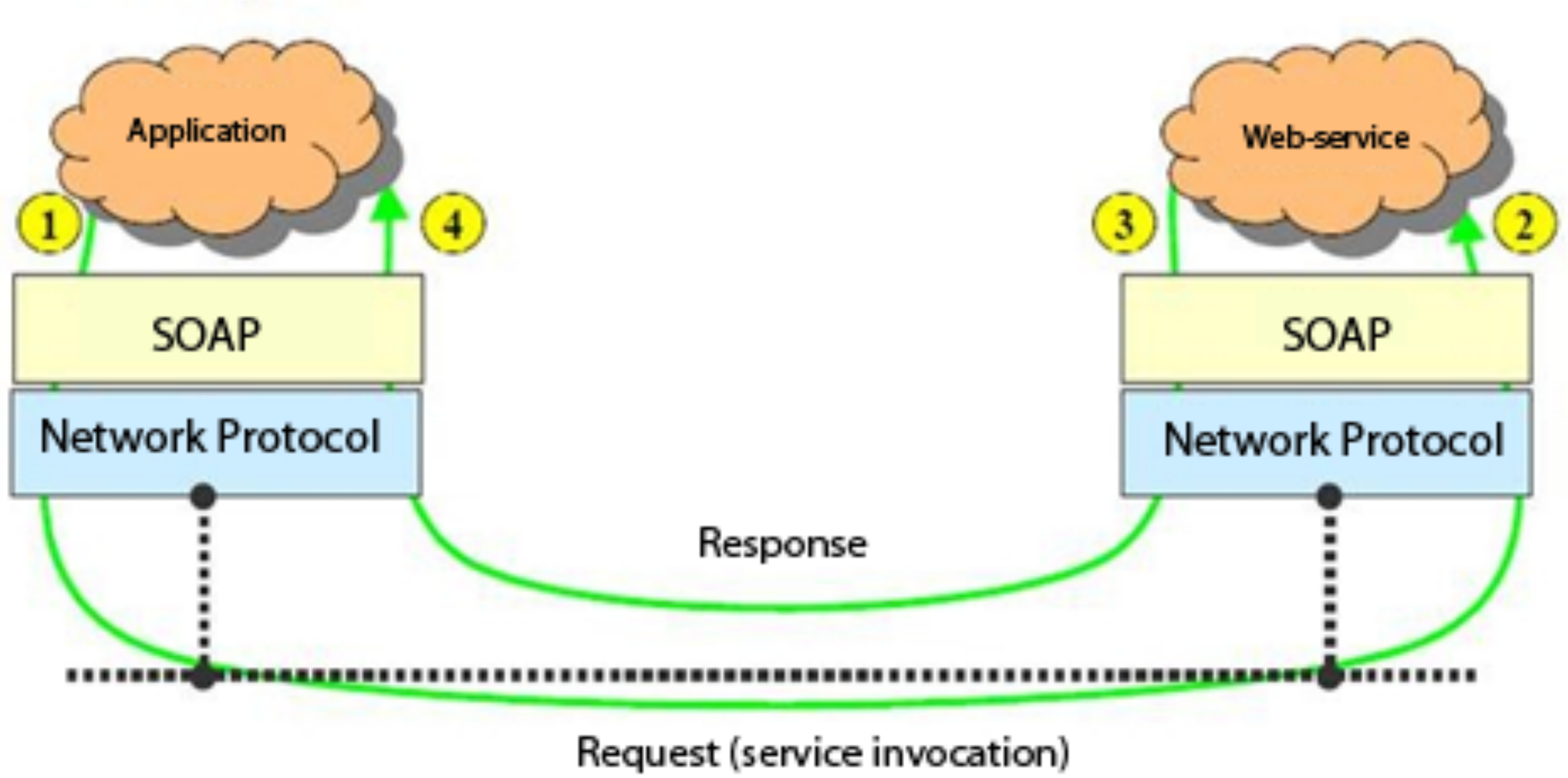
SOAP web-services

- Web-service is a software component
  - Accessible via network
  - Has defined interface
  - Identified by URI (uniform resource identifier)
  - Supports interoperable machine-to-machine interaction over a network

SOAP web-services

- Simple object access protocol
- Maintained by W3C consortium
- Uses xml-based message format
- Platform and language independent
- Can operate over any network protocols (HTTP, TCP, UDP)
- Extensible
- Designed for RPC (remote procedure call) style of web-services

SOAP - Communication



SOAP specification

- Version 1.2 is the latest version
- SOAP messaging framework
  - Rules for processing SOAP message
  - Protocol binding for exchanging SOAP messages between services
  - Structure of SOAP message

Main technologies

- XML
- XML schema - language to define XML document structure, validation of XML document structure
- SOAP
- WSDL (Web-service description language)
  - Web-service interface description

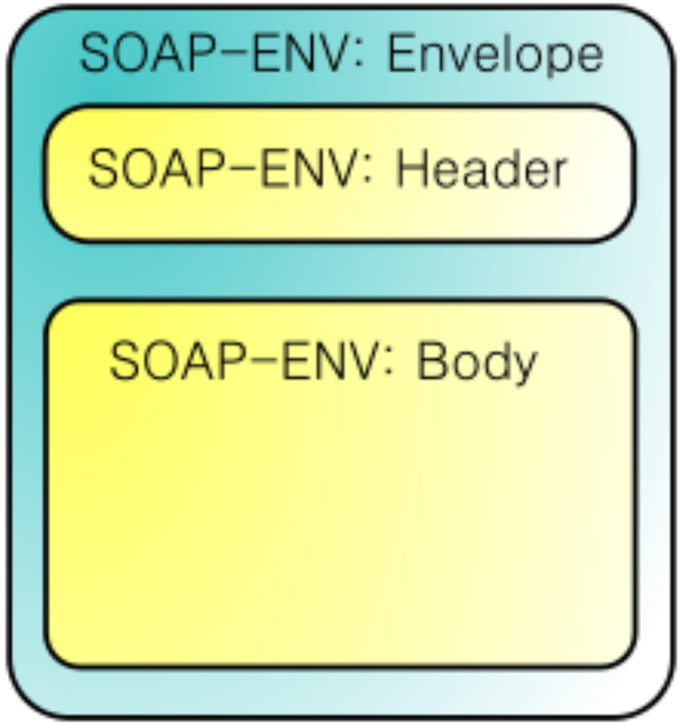
SOAP message

- SOAP message consists of:
  - Envelope
  - Header
  - Body

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:t="http://edu.luxoft.com/ws">

    <env:Header>
    </env:Header>
    <env:Body>
        <t:Now>
            <Date>21 December 2013</Date>
            <Time>18:02:37</Time>
        </t:Now>
    </env:Body>

</env:Envelope>
```



SOAP message: Envelope

- Container for a message
- Structure is described by “Envelope” element defined in namespace:
  - **SOAP 1.1** - http://schemas.xmlsoap.org/soap/
  - **SOAP 1.2** - http://www.w3.org/2003/05/soap-envelope
- Contains:
  - Header (optional)
  - Message body

SOAP message: Header

- The first element in envelope
- Has the following attributes:
  - encodingStyle – element encoding style
  - actor (or role in SOAP 1.2) – logical name of recipient node
  - mustUnderstand – flag if header must be understood on recipient side

SOAP message: Header

- actor attribute contains recipient URI :
  - http://schemas.xmlsoap.org/soap/actor/next - next node
- SOAP 1.2 instead of **actor** supports **role**:
  - http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver
  - http://www.w3.org/2003/05/soap-envelope/role/next
  - http://www.w3.org/2003/05/soap-envelope/role/none

SOAP message: Header

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <tx:Transaction xmlns:tx="http://example.com/transaction"
                        env:role="http://www.w3.org/2003/05/soapenvelope/role/ultimateReceiver"
                        env:mustUnderstand="true">
            5
        </tx:Transaction>
    </env:Header>
    <env:Body>
        <reservation>
            <user>UID-740664012</user>
            <object>OID-83206541</object>
        </reservation>
    </env:Body>
</env:Envelope>
```

Header with mandatory header element

SOAP message: Message body

- Located next to the Header element
- The first element if there is no header element in message

- Contains:
  - Element FAULT defined in SOAP
  - Or arbitrary XML data

SOAP message: Error handling

- Fault contains elements (**SOAP 1.2**):
  - code – that contains:
    - Mandatory element `value` – with error code specified
    - Optional element `subcode` with sub-code *(that contains `value` + `subcode` and so forth)*
  - reason – error description
  - Optional elements:
    - node – URI of node where error occurs
    - role – role of node where error occurs
    - detail – error details in arbitrary XML form

SOAP message: Error handling

```
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:rpc="http://www.w3.org/2002/06/soap-rpc1">
    <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>Processing Error</env:Reason>

            <env:Detail>
                <e:myfaultdetails xmlns:e="http://www.example.org/faults">
                    <message>Name does not match</message>
                    <errorcode>999</errorcode>
                </e:myfaultdetails>
            </env:Detail>

        </env:Fault>

    </env:Body>
</env:Envelope>
```

Code with subcode

Details in arbitrary form

SOAP message: Error handling

- Error codes for **SOAP 1.1**:

  - VersionMismatch – wrong version of SOAP (incorrect nemspace)
  - MustUnderstand – mandatory header element is not understood
  - Client – client side error (ex. wrong arguments passed)
  - Server – server side error (ex. server is out of memory)

SOAP message: Error handling

- Error codes for **SOAP 1.2**:

  - VersionMismatch – wrong version of SOAP (incorrect nemspace)
  - MustUnderstand – mandatory header element is not understood (header Misunderstood should be sent in response message)
  - DataEncodingUnknown – unknown message encoding
  - Sender – client side error (ex. wrong arguments passed)
  - Receiver – server side error (ex. server is out of memory)

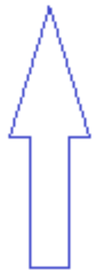
SOAP message: Error handling

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:flt="http://www.w3.org/2002/06/soap-faults">

    <env:Header>
        <tx:Misunderstood qname="xt:Transaction" xmlns:tx="http://sample/transaction"/>
    </env:Header>

    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:MustUnderstand</env:Value>
            </env:Code>
            <env:Reason>
                One or more mandatory headers not understood
            </env:Reason>
        </env:Fault>
    </env:Body>

</env:Envelope>
```



Mandatory header is not understood

SOAP extensions

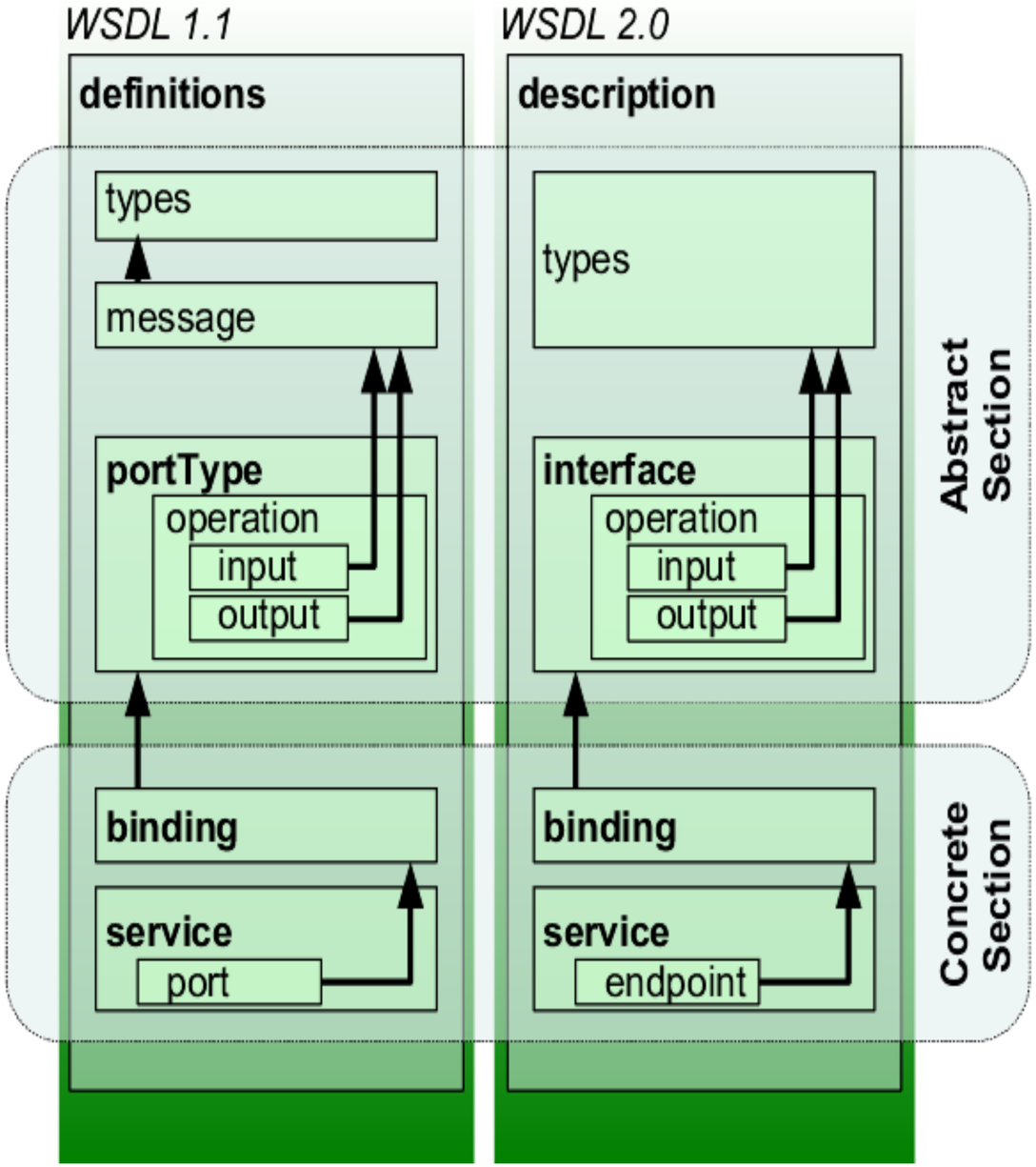
- Main standards extending SOAP protocol:
  - WS-Addressing – defines addressing mechanism independent from transport (address inside SOAP message)
  - WS-Policy – defines mechanism to describe policies restricting or controlling usage of web-service
  - WS-Security – defines mechanism to support web-service security on SOAP level (encryption, digital signing, authentication)
  - WS-Transaction – defines SOAP extension for supporting transactional web-services

WSDL

- Web-service description language
- Versions
  - WSDL 1.1
  - WSDL 2.0
- Platform independent

WSDL

- WSDL describes service on the 2 levels:
  - Abstract *(types, messages, operations)*
  - Concrete *(bindings, endpoints)*



WSDL

- Steps to describe web-service with WSDL:

    - Abstracts:
        - Defines types (using XSD)
        - Define messages and its structures
        - Define operations and its in/out messages
    - Concretes:
        - Bind abstract operations to SOAP protocol and transport protocol
        - Define web-service endpoint

WSDL 1.1

```
<wsdl:definitions targetNamespace="http://greeting/" xmlns:tns="http://greeting/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <wsdl:types>
        <xs:schema targetNamespace="http://greeting/" xmlns:tns="http://greeting/"...>
    </wsdl:types>
    <wsdl:message name="getHello">
        <wsdl:part name="parameters" element="tns:getHello"/>
    </wsdl:message>
    <wsdl:message name="getHelloResponse">
        <wsdl:part name="parameters" element="tns:getHelloResponse"/>
    </wsdl:message>
    <wsdl:portType name="Greeting">
        <wsdl:operation name="getHello">
            <wsdl:input name="getHello" message="tns:getHello"/>
            <wsdl:output name="getHelloResponse" message="tns:getHelloResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="GreetingServiceSoapBinding" type="tns:Greeting">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="getHello"...>
    </wsdl:binding>
    <wsdl:service name="GreetingService">
        <wsdl:port name="GreetingPort" binding="tns:GreetingServiceSoapBinding"...>
    </wsdl:service>
</wsdl:definitions>
```

1. Define types
2. Define messages
3. Define operations
4. Bind to SOAP and HTTP
5. Define endpoint

WSDL - usage

- WSDL document defines a web-service and provides to client information about how to use it

  - Web-service – implements interface defined in WSDL
  - Client – communicates with web-service sending messages that are build according to WSDL

.

WSDL 1.1

- Definitions is a root element containing all web-service definitions
- 6 main elements:
  - Types – contains types definition
  - Message – describes an abstract message
  - PortType – describes operations supported by service
  - Binding – contains mapping rules to SOAP and HTTP
  - Port – defines web-service endpoint
  - Service – container for port elements

WSDL 1.1

- Message – defines abstract message *(that will be bind to SOAP or another protocol message)*

- Has unique name (`name` attribute)
- Consists of parts (`part` element)
  Every part of a message can be bind to SOAP message body or header

- Type of a message part can be specified:
  - By reference to abstract type (section `Types`)
  - By reference to element (section `Types`)

WSDL 1.1

```
<wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://greeting/" xmlns:tns="http://greeting/">

        <xs:complexType name="AuthDetails">
            <xs:sequence>
                <xs:element name="user" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>

        <xs:complexType name="Hello">
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>

        <xs:element name="HelloElement" type="tns:Hello"/>
    </xs:schema>
</wsdl:types>
```

Types definition

```
<wsdl:message name="getHello">
    <wsdl:part name="hello" element="tns:HelloElement"/>
    <wsdl:part name="auth" type="tns:AuthDetails"/>
</wsdl:message>
```

Message part wit reference to element

Message part wit reference to type

WSDL 1.1

- PortType defines web-service interface as a set of named operations
- Has unique name (`name` attribute)
- Contains operations definition (`operation` element)

- Operation messages defined by:
  - Input Message – `input` element
  - Output Message – `output` element
  - Fault Message – `fault` element

WSDL 1.1

- Message exchange patters:

  - One-Way – operation has only input message
    Web-service receives message but does not response with any message

  - Request-Response – operation has input and output (or fault) messages definition
    Web-service receives message and responses with response or fault message

WSDL 1.1

```
<wsdl:message name="getHello">
    <wsdl:part name="parameters" element="tns:getHello"/>
</wsdl:message>

<wsdl:message name="getHelloResponse">
    <wsdl:part name="parameters" element="tns:getHelloResponse"/>
</wsdl:message>

<wsdl:message name="getHelloFault"/>

<wsdl:portType name="Greeting">
    <wsdl:operation name="getHello">
        <wsdl:input name="getHello" message="tns:getHello"/>
        <wsdl:output name="getHelloResponse" message="tns:getHelloResponse"/>
        <wsdl:fault name="getHelloFault" message="tns:getHelloFault"/>
    </wsdl:operation>
</wsdl:portType>
```

WSDL 1.1

- Binding – makes an abstract web-service interface real (PortType)

- Defines:
  - Transport protocol (ex. HTTP)
  - Binding messages parts to SOAP message body and headers
  - Version of SOAP protocol
  - Which binding style to use (document/literal)

WSDL 1.1

```
<wsdl:binding name="GreetingServiceSoapBinding" type="tns:Greeting">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getHello">
        <soap:operation soapAction="getHello" style="document"/>
        <wsdl:input name="getHello">
            <soap:header use="literal" message="getHello" part="parameters"/>
        </wsdl:input>
        <wsdl:output name="getHelloResponse">
            <soap:body use="literal"/>
        </wsdl:output>
        <wsdl:fault name="getHelloFault">
            <soap:body use="literal"/>
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
```

Use SOAP over HTTP

IN message part "parameters" is mapped to SOAP header

OUT and FAULT messages are mapped to SOAP body

WSDL 1.1

- Port – defines real endpoint for particular Binding
  - name attribute – name of port
  - binding attribute – reference to Binding

- Service – contains Port definitions

```
<wsdl:service name="GreetingService">
    <wsdl:port name="GreetingPort" binding="tns:GreetingServiceSoapBinding">
        <soap:address location="http://localhost:8080/GreetingPort"/>
    </wsdl:port>
</wsdl:service>
```

SOAP binding styles

- How message is translated to XML depends on:
  - Binding Style (element of `<soap:binding>`) – `Document` or `RPC`
  - Use (attribute of `<soap:header>` and `<soap:body>`) – `literal` or `encoded`

- Possible combinations:
  - RPC/encoded
  - RPC/literal
  - Document/literal + Document/literal Wrapped

RPC/encoded

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

<wsdl:message name="empty"/>

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

web-service
interface definition

SOAP message for
this interface

```
<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>
```

RPC/literal

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

<wsdl:message name="empty"/>

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

web-service
interface definition

SOAP message for
this interface

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

Document/literal

```
<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>
```

web-service interface definition

SOAP message for this interface

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

Java API for web-services: JAX WS

- JAX-WS is a Java programming language API for creating web services
  - Defines a standard Java-to-WSDL mapping *(JSR224)*
  - Part of Java SE (Java 6 - Java 10)
  - Supersedes JAX-RPC

- Provides ability to create web-services:
  - RPC-oriented (like JAX-RPC)
  - Message-oriented

Java API for web-services: JAX WS

- Frameworks supporting JAX-WS:
  - GlassFish Metro (JAX-WS reference implementation)
  - Apache Axis 2
  - Apache CXF

Java API for web-services: JAX WS

- JAX-WS supports the following approached:
  - Top-Down

    Web-service development starts from WSDL. The `wsimport` utility is used for source code generation.

  - Bottom-Up

    Web-service development starts from service endpoint interface (SEI) definition in Java. The `wsgen` utility can be used for generation of needed artifacts.

JAX WS - top down

- WSDL namespace is mapped to Java package name (targetNamespace attribute of <wsdl:definitions>)
- Port name is mapped to name of SEI (name attribute of <wsdl:portType>)
- Operations are mapped to methods of SEI (name attributes of <wsdl:operation>)
- Messages are mapped to SEI method parameters and return value (<wsdl:input> and <wsdl:output>)
- Fault messages are mapped to Java exceptions of SEI methods <wsdl:fault>

JAX WS - bottom-up

- SEI can be defined:
  - Explicitly – through Java interface definition
  - Implicitly – deriving from web-service implementation class

- SEI (or implementation class) should have one of the following annotations:
  - @WebService – RPC oriented web-service
  - @WebServiceProvider – Message oriented web-service

JAX WS - bottom-up

- Web-service implementation class should:
  - Not be `abstract` or `final`
  - Have `public` constructor with no parameters
  - Not have `finalize` methods
  - Have annotation `@WebService` or `@WebServiceProvider`

- You can use `@PreDestroy` and `@PostConstruct` annotations to manage web-service implementation instance lifecycle

JAX WS - bottom-up

```java
package examples;

@WebService(targetNamespace = "http://agerman/ws/examples")
public interface HelloService {
    String sayHello(String name, int age) throws HelloException;
}
```

```java
package examples;

@WebService(endpointInterface = "examples.HelloService")
public class HelloServiceImpl implements HelloService {
    public String sayHello(String name, int age) throws HelloException {
        return "Hello " + name + "!";
    }
}
```

Explicit SEI definition

```java
package examples;
@WebService(targetNamespace = "http://agerman/ws/examples")
public class HelloServiceImpl {
    public String sayHello(String name, int age) throws HelloException {
        return "Hello " + name + "!";
    }
}
```

SEI derived from web-servic class

JAX WS - bottom-up - wsdl

```
<definitions targetNamespace="http://agerman/ws/examples"
             xmlns:tns="http://agerman/ws/examples"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        ...
    </types>

    <message name="sayHello">
        <part name="parameters" element="tns:sayHello"/>
    </message>
    <message name="sayHelloResponse">
        <part name="parameters" element="tns:sayHelloResponse"/>
    </message>
    <message name="HelloException">
        <part name="fault" element="tns:HelloException"/>
    </message>

    <portType name="HelloService">
        <operation name="sayHello">
            <input message="tns:sayHello"/>
            <output message="tns:sayHelloResponse"/>
            <fault message="tns:HelloException" name="HelloException"/>
        </operation>
    </portType>

</definitions>
```

JAX WS - bottom-up

- @WebService: annotation parameters
  - name – name of <wsdl:portType>
  - targetNamespace – namespace
  - serviceName – name of <wsdl:service>
  - portName – name of <wsdl:port> inside the <wsdl:service>
  - wsdlLocation – URI to WSDL document
  - endpointInterface – full name of SEI (*package name* + *interface name*)

JAX WS - bottom-up

- SEI methods:
  - Have annotation `@WebMethod` (optional)
  - Should be `public`
  - Should not be `static` or `final`

- SEI methods parameters:
  - Have annotation `@WebParam` (optional)
  - Should be JAXB-compatibale
  - Use type Holder for OUT or IN/OUT parameters

JAX WS - bottom-up

- @WebMethod annotation parameters:
  - operationName – name of <wsdl:operation>
  - action – value of HTTP header SOAPAction
- @WebParam annotation parameters:
  - partName – name of <wsdl:part>
  - header – flag to map parameter to SOAP header
  - mode – parameter direction mode:
    - WebParam.Mode.IN
    - WebParam.Mode.OUT
    - WebParam.Mode.INOUT

JAX WS - bottom-up

```java
@WebMethod(action = "sayHello", operationName = "sayHelloOperation")
String sayHello(
        @WebParam(partName = "name_param") String name,
        int age) throws HelloException;
```

```xml
<definitions targetNamespace="http://examples/"
             xmlns:tns="http://examples/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <portType name="HelloService">
        <operation name="sayHelloOperation">
            <input message="tns:sayHelloOperation"/>
            <output message="tns:sayHelloOperationResponse"/>
            <fault message="tns:HelloException" name="HelloException"/>
        </operation>
    </portType>

    <binding name="HelloServiceImplPortBinding" type="tns:HelloService">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="sayHelloOperation">
            <soap:operation soapAction="sayHello"/>
            <input> <soap:body use="literal"/> </input>
            <output> <soap:body use="literal"/> </output>
            <fault name="HelloException"> <soap:fault name="HelloException" use="literal"/> </fault>
        </operation>
    </binding>
</definitions>
```

JAX WS - bottom-up

```java
@WebService(name = "HelloService", targetNamespace = "http://hello.ws/")
public interface HelloService {
    @WebMethod
    @RequestWrapper(localName = "sayHello", className = "ws.hello_client.SayHello")
    @ResponseWrapper(localName = "sayHelloResponse", className = "ws.hello_client.SayHelloResponse")
    public String sayHello(
        @WebParam String arg0,
        @WebParam int arg1) throws HelloServiceException;
}
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name="sayHello", propOrder={"arg0","arg1"})
public class SayHello {
    protected String arg0;
    protected int arg1;

    public String getArg0() {
        return arg0;
    }
    public void setArg0(String value) {
        this.arg0 = value;
    }
    public int getArg1() {
        return arg1;
    }
    public void setArg1(int value) {
        this.arg1 = value;
    }
}
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(
        name = "sayHelloResponse",
        propOrder = {"_return"})
public class SayHelloResponse {
    @XmlElement(name = "return")
    protected String _return;

    public String getReturn() {
        return _return;
    }
    public void setReturn(String value) {
        this._return = value;
    }
}
```

JAX WS - bottom-up

- JAX-WS supports web-services:

- RPC oriented:
  - SEI has annotation @WebService
  - Operates with domain objects

- Message oriented:
  - SEI has annotation @WebServiceProvider
  - Operates with messages

JAX WS - calling web services

- javax.xml.ws.Service – acts as factory of objects needed for calling web-service:

- Creates **Proxy-objects** for communications with RPC-oriented web-services

- Creates **Dispatch- objects** for communications with Message-oriented web-services

- Created by `static` factory method:

- Service.create(URL wsdlUrl, QName serviceName)

JAX WS - calling web services

- javax.xml.ws.Service methods:

- getPort() – creates proxy that implements SEI
- createDispatch() - creates Dispatch-object
- setHandlerResolver() – defines strategy to find handlers (HandlerResolver)
- setExecutor() – defines instance of java.util.concurrent.Executor for calling web-services asynchronously

- Class `Service` is also super class for all `*Service` classes produced by `wsimport` utility

JAX WS - calling web services

- Message-oriented client:
- Defined by `javax.xml.ws.Dispatch`
- Created with `createDispatch()` of the `javax.xml.ws.Service` object
- Supports modes:
  - Service.Mode.MESSAGE
  - Service.Mode.PAYLOAD
- Supports sources to get message content:
  - javax.xml.transform.Source
  - javax.xml.soap.SOAPMessage
  - javax.activation.DataSource

JAX WS - calling web services

```
String requestPayloadText =
            "<ns2:sayHello xmlns:ns2='http://hello.ws/'>" +
                "<arg0>John</arg0>" +
                "<arg1>25</arg1>" +
            "</ns2:sayHello>";

Source requestPayload = new SAXSource(
        new InputSource(
                new ByteArrayInputStream(requestPayloadText.getBytes())));

HelloServiceImplService helloService = new HelloServiceImplService();
Dispatch<Source> dispatch = helloService.createDispatch(
        new QName("http://hello.ws/", "HelloServiceImplPort"),
        Source.class,
        Service.Mode.PAYLOAD);


Source responsePayload = dispatch.invoke(requestPayload);
```

Content of SOAP body for request message

Create Dispatch object with PAYLOAD mode

Call web-service and receive response message