Woldia University
School of Computing

Department of Software Engineering
Course Title: **Web Service,**
Course Code: **SEng5127**

Chapter Two: Defining SOAP Message Using WSDL

By : Demeke G.
AY:2018 E.C

Outline

- ☛ XML (**eXtensible Markup Language**) Essentials
- ☛ Structure of SOAP Messages
- ☛ Anatomy of a WSDL Document

Learning Outcome:

- ✓Understand XML Fundamentals
- ✓Explain the Structure and Role of SOAP
- ✓Analyze and Construct WSDL Documents
- ✓Integrate XML, SOAP, and WSDL Concepts

Introduction to XML

✓XML is a markup language used to store and transport data.

✓It focuses on data representation rather than presentation.

✓Human-readable and machine-readable.

✓Platform and language independent.

✓Uses **tags** to describe data (**user-defined**).

✓Supports **nested elements** and **hierarchical** structures.

✓consists of a start **tag**, **content**, and an **end tag.**

✓Elements can contain text, attributes, other elements, or be empty.

✓Empty elements can also be written in a self-closing way: <address />

- Example

```
<Student>
  <Name>John Doe</Name>
  <Age>22</Age>
  <Department>Computer Science</Department>
</Student>
```

✓An attribute provides additional information about an element.

✓Use comment like <!-- This is a comment -->

✓XML does not truncate multiple white-spaces

XML vs HTML

Feature	XML	HTML
Purpose	Data storage and transfer	Data presentation
Tag Definition	User-defined	Predefined
Error Handling	Strict	Lenient
Case Sensitivity	Case Sensitivity	Non Case Sensitivity
Data Type Support	Can define types using XSD	No data typing

DTD vs XSD

- Both are used to define the legal building blocks and structure of an XML document.
- **Document Type Definition (DTD)**
  - The older standard for XML validation.
  - Defines elements, attributes, and their relationships.
  - **Limitations:**
    - No support for data types (e.g., string, integer, date).
    - Syntax is not XML-based, making it less flexible.
- **XML Schema Definition (XSD)**
  - The modern and more powerful alternative to DTD.
  - XSD is used to define, describe, and validate the structure and content of XML documents.
  - uses a set of predefined tags and attributes to define the structure, content, and data types
  - These tags are part of the http://www.w3.org/2001/XMLSchema namespace
  - written in XML syntax itself and supports data types, namespaces, and validation rules.
  - Supports a rich set of data types (e.g., <xs:int>, <xs:date>).
  - it can be parsed and manipulated like any other XML document.
  - Supports namespaces.

Cont..

Structural tags

- **<xs:schema>**: The root element of every XML schema. It defines the target namespace for the schema and contains all other declarations and definitions.
- **<xs:element>**: Declares an element that can appear in the XML document. It defines the name and data type of the element.
  - **Attributes:**
    - name: The name of the element.
    - type: The data type of the element (e.g., xs:string, xs:int).
- **<xs:attribute>**: Declares an attribute that can appear within an element.
  - **Attributes:**
    - name: The name of the attribute.
    - type: The data type of the attribute.
    - use: Specifies if the attribute is optional or required

Cont..

Type definition tags

- These tags allow you to define both simple and complex data types for elements and attributes
- **<xs:complexType>:** Defines a complex type for an element that contains other elements and/or attributes.
- **<xs:simpleType>:** Defines a simple type for an element or attribute that contains only text content.
- **<xs:restriction>:** Restricts an existing data type with facets (e.g., maxLength, pattern).
- **<xs:sequence>:** Specifies that the child elements must appear in a specific, predefined order.
- **<xs:choice>:** Specifies that one of the child elements can be chosen from a group.
- **<xs:all>:** Specifies that the child elements can appear in any order, but each must appear at most once.

Examples

- Used when an element contains **other elements or attr**

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
```

- **Used to define a custom simple type with restrictions**

```
<xs:simpleType name="ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
```

- Allows **only one element** from a list to appear.

```
<xs:choice>
  <xs:element name="email" type="xs:string"/>
  <xs:element name="phone" type="xs:string"/>
</xs:choice>
```

```
<xs:restriction base="xs:string">
  <xs:enumeration value="Male"/>
  <xs:enumeration value="Female"/>
</xs:restriction>
```

Sets **numeric range limits**

```
<xs:minInclusive value="18"/>
<xs:maxInclusive value="60"/>
```

Cont..

Built-in simple data types

- XSD comes with many built-in simple types that can be used to define an element's or attribute's content.
- **Strings**: xs:string, xs:normalizedString, xs:token
- **Numbers**: xs:decimal, xs:integer, xs:int, xs:long, xs:float, xs:double
- **Dates and times**: xs:date, xs:time, xs:dateTime
- **Other types**: xs:boolean, xs:anyURI

Grouping and modularization tags

- **<xs:import>**: Allows the use of schema components from a different namespace.
- **<xs:include>**: Incorporates schema components from another schema with the same target namespace.

Sample XML data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<studentRegistration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="student.xsd">
  <student id="STU001">
    <firstName>John</firstName>
    <lastName>Maki</lastName>
    <gender>Male</gender>
    <age>23</age>
    <contact>
      <email>jonmaki@example.com</email>
      <phone>0912345678</phone>
    </contact>
    <department>Software Engineering</department>
    <status>Active</status>
  </student>
</studentRegistration>
```

Corresponding XSD Schema (student.xsd)(1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Root element -->
 <xs:element name="studentRegistration">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="student" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <!-- Basic info -->
       <xs:element name="firstName" type="xs:string"/>
       <xs:element name="lastName" type="xs:string"/>
       <!-- Gender with restriction -->
       <xs:element name="gender">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Male"/>
          <xs:enumeration value="Female"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
```

```
<!-- Age with numeric restriction -->
       <xs:element name="age">
        <xs:simpleType>
         <xs:restriction base="xs:integer">
          <xs:minInclusive value="15"/>
          <xs:maxInclusive value="60"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
       <!-- Contact info (choice example) -->
       <xs:element name="contact">
        <xs:complexType>
         <xs:choice>
          <xs:element name="email" type="xs:string"/>
          <xs:element name="phone" type="xs:string"/>
         </xs:choice>
        </xs:complexType>
       </xs:element>
```

Corresponding XSD Schema (student.xsd)(2)

```
 <xs:element name="department" type="xs:string"/>
<xs:element name="status">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Active"/>
          <xs:enumeration value="Inactive"/>
          <xs:enumeration value="Graduated"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
 </xs:sequence>
<xs:attribute name="id" type="xs:string"
use="required"/>
```

```
</xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

XML Parsing

- Parsing is the process of reading XML documents and extracting information.
- **Parser** is a software library or program that reads an XML document and provides an interface for accessing its content and structure.
- **JAXB (Java Architecture for XML Binding)**
  - Specifically for the Java programming language.
  - Automatically generates Java classes from an XSD and vice-versa.
  - Simplifies marshaling and unmarshaling
- **Marshalling**: The process of converting Java objects into an XML document.
- **Unmarshalling**: The process of converting an XML document into a corresponding set of Java objects.
- **Benefits of Binding APIs:**
  - Eliminates boilerplate code for parsing.
  - Allows developers to work with familiar objects rather than raw XML.
  - Improves developer productivity and reduces errors.

Cont..

- JAXB uses annotations or an XML schema to define how Java classes and their properties map to XML elements and attributes.
- JAXB reads the XML and populates the Java objects
- JAXB can generate Java classes directly from an XML Schema Definition (XSD), providing a strongly typed representation of the XML structure.
- Developers can annotate Java classes and fields with JAXB annotations (e.g., **@XmlRootElement**, **@XmlElement**, **@XmlAttribute**) to control the mapping between Java objects and XML.
- To use JAXB in newer Java versions, need to explicitly add the necessary dependencies (e.g., jakarta.xml.bind-api and jaxb-impl)

How JAXB works

- Developers can use JAXB in two primary ways:
- **From XML Schema (XSD):** Use the **xjc** schema compiler tool to automatically generate a set of Java classes from an XML Schema Definition (XSD) file.
  - Generated classes include JAXB annotations that define the mapping.
- **From Java Classes:** Use annotations directly on existing Java classes to define how they should be mapped to XML.
- The **schemagen** tool can then generate an XML schema from these annotated classes.
- JAXB uses a set of annotations to control the binding process.

  @XmlRootElement: Designates the root element of the XML document.

  @XmlElement: Maps a Java property to an XML element.

  @XmlAttribute: Maps a Java property to an XML attribute.

  @XmlTransient: Instructs JAXB to ignore a specific Java property during marshalling.

  @XmlType: Allows control over the ordering of elements in the XML output.

XML Namespaces

- A mechanism to avoid element name conflicts when combining XML documents from different vocabularies.
- If two XML documents both define a <title> element, a parser won't know which one you mean when combining them. (**problem**)
  - Namespaces use a URI (Uniform Resource Identifier) to uniquely identify a set of element and attribute names. (**solution**)
- **Example:** <h:table> vs <f:table>, where h and f are namespace prefixes.
- A namespace is declared using the **xmlns** attribute.
- The **URI** doesn't need to point to an actual file; it just needs to be a unique identifier.
- **targetNamespace** is an attribute in XSD that defines a unique namespace for all the elements and types declared in that schema.
- **elementFormDefaul**t="qualified/unqualified“: an attribute of the <xs:schema> element
  - **Qualified**: All local elements (inside complex types) must use the target namespace i.e. <std:name>
  - **unqualified** (default):Only global elements (top-level ones) use the namespace. Local ones do not. i.e. <name>

Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://example.com/student"
      xmlns:std="http://example.com/student"
      elementFormDefault="qualified">
 <xs:element name="student">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Structure of a SOAP Message

- SOAP is an XML-based protocol used to exchange structured information in web services over the Internet.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
  <!-- Optional metadata (e.g., authentication, routing info) -->
 </soap:Header>
 <soap:Body> <!-- Actual message or method call -->
<m:GetStudentDetails xmlns:m="http://example.com/student">
    <m:studentId>1001</m:studentId>
  </m:GetStudentDetails>
 </soap:Body>
 <soap:Fault>
  <!-- contains error details if processing fails -->
 </soap:Fault>
</soap:Envelope>
```

```
<auth:Authentication
xmlns:auth="http://example.org/auth">
<auth:Username>alexander</auth:Username>
    <auth:Password>12345</auth:Password>
   </auth:Authentication>
```

```
<soap:Fault xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Code>
  <soap:Value>soap:Sender</soap:Value>
 </soap:Code>
 <soap:Reason>
  <soap:Text xml:lang="en">Invalid Student</soap:Text>
 </soap:Reason>
 <soap:Detail>
  <errorcode>STUD-404</errorcode>
  <errordescription>The Student ID does not exist</errordescription>
 </soap:Detail>
</soap:Fault>
```

Cont..

- Envelope: Root element that defines the start and end of the SOAP message.
- Header: Optional element that contains additional information (security, transaction data).
- Body: Contains the actual data or function call sent to the web service.
- Fault: Optional element used to report errors that occur while processing the message.

  <soap:Fault> :Root element for SOAP error messages

  <soap:Code>: Contains the fault code (e.g., soap:Sender, soap:Receiver)

  <soap:Reason>: Explains why the fault occurred (human-readable text)

  <soap:Detail>: Provides application-specific details like error codes, messages, etc.

Operations, Messages, and Faults

- SOAP messages represent **operations** defined by the web service.
- Operations are the actions or methods provided by the web service.
- Operations Defined in WSDL under <portType> and invoked using SOAP requests.
- Example:

```
<soap:Body>
  <m:getStudentDetails>
    <m:studentId>1001</m:studentId>
  </m:getStudentDetails>
</soap:Body>
```

  getStudentDetails is an Operation

- A **message** represents the **data exchanged** between client and service
- There are Two main types:
  - Request Message → sent from client to server.
  - Response Message → returned by the server.

Cont..

```
<soap:Body>
  <m:getStudentDetailsResponse>
    <m:name> Alemu </m:name>
    <m:age> 23 </m:age>
  </m:getStudentDetailsResponse>
</soap:Body>
```

- SOAP faults report errors during message processing.
- SOAP faults is special type of message returned in the body to indicate an error.
- Contained inside the **<soap:Fault>** element.

```
      <soap:Fault>
        <faultcode>soap:Client</faultcode>
        <faultstring>Invalid student ID</faultstring>
      </soap:Fault>
```

Role of SOAP in Web Services

- SOAP provides a common messaging framework for client–server communication in a distributed environment.
- SOAP messages are encoded in XML, ensuring interoperability between systems regardless of platform or language.
- SOAP can use different transport protocols such as **HTTP**, **SMTP**, **JMS**, or **TCP/IP**.
- Additional features like security, transactions, or routing can be added using **SOAP headers**.
- SOAP defines a standard error handling mechanism using the **Fault** element, making communication reliable.

Anatomy of a SOAP Message

- An XML document with a specific format.
- Composed of four main elements:
  - **Envelope:** The root element, mandatory for every SOAP message, which defines the start and end of the message.
  - **Header:** An optional element for carrying application-specific information like authentication or routing.
  - **Body:** A mandatory element that contains the actual payload, such as a request or response.
  - **Fault:** An optional element inside the Body, used for reporting errors.

Defining Interfaces with WSDL

- An XML-based language that describes the functionality of a web service in a machine-readable format.
- The "**interface**" defines the abstract, reusable parts of a web service.
- It describes the operations that are available, the data types they use, and the format of the messages exchanged.
- Acts as a **contract** between the service provider and the consumer.
- Interface definition contains three main parts:
  - **<types>:** Defines the data types used in the messages, typically via XML Schema (XSD).
  - **<message>:** An abstract definition of the data being communicated. There is a message for the input parameters and another for the output parameters of an operation.
  - **<portType> / <interface>:** An abstract set of operations supported by the service. It groups related messages to define a complete request-response action.

Example 1:

```
<types>
    <xsd:schema targetNamespace="http://example.com/calculator">
      <xsd:element name="addNumbersRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="a" type="xsd:int"/>
            <xsd:element name="b" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="addNumbersResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="result" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>
```

```
<message name="AddNumbersInput">
    <part name="parameters"
element="tns:addNumbersRequest"/>
  </message>
  <message name="AddNumbersOutput">
    <part name="parameters"
element="tns:addNumbersResponse"/>
  </message>
  <!-- 3. Port Type (Interface Definition) -->
  <portType name="CalculatorPortType">
    <operation name="addNumbers">
      <input message="tns:AddNumbersInput"/>
      <output message="tns:AddNumbersOutput"/>
    </operation>
  </portType>
```

Specifying Implementation with WSDL

- The "**implementation**" section takes the abstract interface and adds concrete details about how the service is physically accessed.
- The second half of a **WSDL** document specifies how to access the abstract interface.
- <**binding**>: Defines the concrete protocol and data format for a particular port type.
  - Specifies details like the transport protocol (e.g., SOAP over HTTP) and the SOAP message style (RPC or Document).
- <**port**> / <endpoint>: Defines a single communication endpoint by associating a binding with a network address (URI).
- <**service**>: A collection of related ports. It groups all the endpoints that make up the complete web service.

Example 1:

```
<binding name="CalculatorBinding"
type="tns:CalculatorPortType">
        <soap:binding style="document"
transport=
"http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addNumbers">
            <soap:operation soapAction =
"http://example.com/calculator/addNumber
s"/>
 <input><soap:body use="literal"/>   </input>
  <output> <soap:body use="literal"/> </output>
  </operation>
    </binding>
```

```
 <service name="CalculatorService">
   <port name="CalculatorPort"
binding="tns:CalculatorBinding">
 <soap:address
location="http://localhost:8080/calculat
or"/>
</port>
   </service>
```

Example 2:

```
<definitions name="StudentService"
         targetNamespace="http://example.com/student"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:tns="http://example.com/student">
  <types>
   <xsd:schema targetNamespace="http://example.com/student">
    <xsd:element name="GetStudentRequest" type="xsd:string"/>
    <xsd:element name="GetStudentResponse" type="xsd:string"/>
   </xsd:schema>
 </types>
```

```
     <!-- Messages -->
     <message name=
     "GetStudentRequestMessage">
       <part name="parameters"
     element="tns:GetStudentRequest"/>
      </message>
<message
name="GetStudentResponseMessage">
   <part name="parameters"
element="tns:GetStudentResponse"/>
  </message>
  <!-- Operations -->
  <portType name="StudentPortType">
   <operation name="getStudentDetails">
     <input
message="tns:GetStudentRequestMessage"/>
     <output
message="tns:GetStudentResponseMessage"/>
   </operation>
  </portType>
</definitions>
```

Example 2 con..

```
<binding name="StudentServiceSoapBinding“
 type="tns:StudentPortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getStudentDetails">
      <soap:operation
 soapAction="http://example.com/student/getStudentDetails"
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
```

```
 <service name="StudentService">
    <documentation>Service to retrieve
student details</documentation>
    <port name="StudentPort"
binding="tns:StudentServiceSoapBinding
">
<soap:address
location="http://localhost:8080/
StudentService"/>
    </port>
  </service>

</definitions>
```

Conclusion

- WSDL defines how SOAP-based web services communicate.
- It specifies the structure of messages, operations, and endpoints.
- Using WSDL, SOAP messages are described clearly with their input and output data.
- This ensures consistent communication between clients and servers.
- WSDL provides interoperability, standardization, and easier integration across platforms.