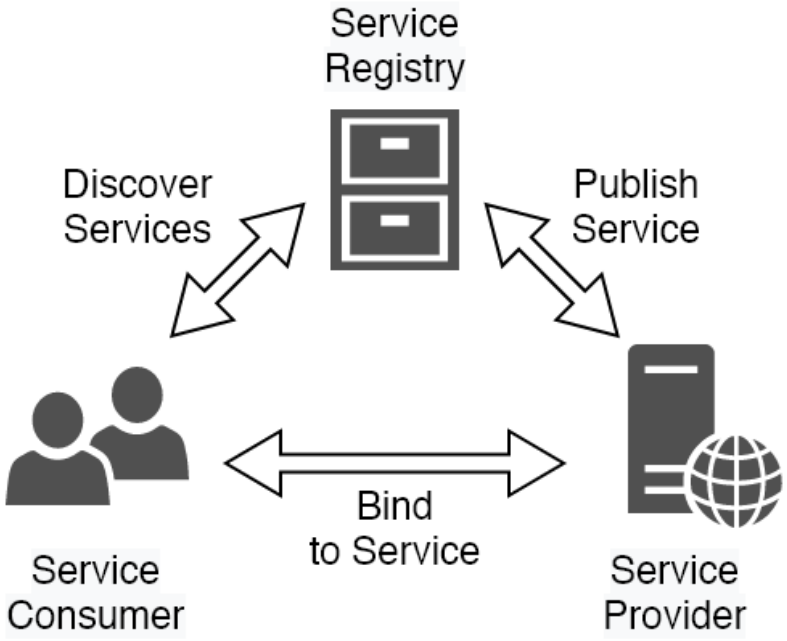
Information systems modeling

Tomasz Kubik

SOA (Service-Oriented Architecture)

- Basic scenario
  - a client discovers a web service in the network and analyses its description provided in the corresponding WSDL document,
  - a client learns from the description about operations provided and data types used by the service,
  - a client decides, which operations he or she is interested in, and builds the application code for it
  - if a client wants and can, it invokes chosen service's operations.



WSDL - Web Services Description Language

- an XML application, developed by W3C, used to write a formal, technical description of web service interfaces.
- provides distinct definitions and terminologies used in web service description
- a schema for services declarations as collections of network endpoints, or ports, offering operations, and exchanging data through messages
- separates abstract definitions from concrete use or instance
- enough to generate code partially, for a client or a server side of the service, in an automatic manner (it is also possible to generate WSDL description on a base of existing code of web service implementation).

WSDL versioning

- Two major versions of WSDL specification
  - WSDL 1.1 (old W3C proposition, but still used),
  - WSLD 2.0 (current W3C recommendation)
- The WSDL 2.0 got its name after renaming WSDL 1.2 because of some substantial differences between 1.1 and 1.2 versions
- WSDL 2.0 accepts binding to all the HTTP request methods (not only GET and POST as in WSDL 1.1) so it better suits for RESTful web services implementation.
- There is also SOAP 1.1 binding recommendation available. However, WSDL 1.1 has better support from software development tools.

Facts

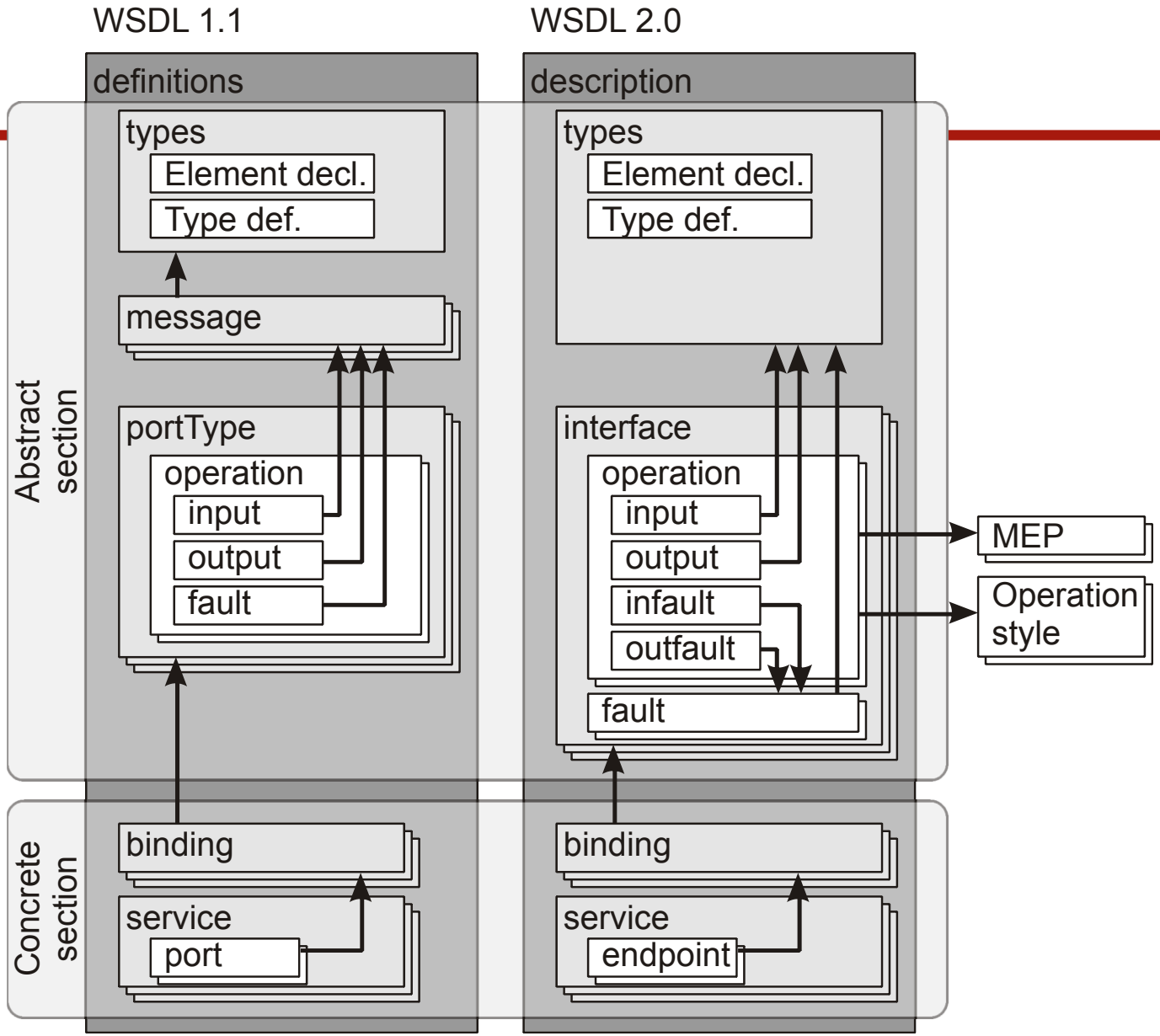
Specification	Publication date	Status
WSDL 1.1 Note	15 Mar 2001	Draft /Proposal
WSDL Usage Scenarios	04 Jun 2002	Draft /Proposal
WSDL Requirements	28 Oct 2002	Draft /Proposal
WSDL Architecture	11 Feb 2004	Draft /Proposal
WSDL Glossary	11 Feb 2004	Draft /Proposal
WSDL Usage Scenarios	11 Feb 2004	Draft /Proposal
WSDL 1.2 Core Language	11 Jun 2003	Draft /Proposal
WSDL 1.2 Message Patterns	11 Jun 2003	Draft /Proposal
WSDL 1.2 Bindings	11 Jun 2003	Draft /Proposal
WSDL 2.0 Primer	26 Jun 2007	Recommendation
WSDL 2.0 Core Language	26 Jun 2007	Recommendation
WSDL 2.0 Adjuncts	26 Jun 2007	Recommendation
WSDL 2.0 SOAP 1.1 Binding	26 Jun 2007	Recommendation
WSDL 2.0 RDF Mapping	26 Jun 2007	Recommendation
Web Services Addressing Core	09 May 2006	Recommendation
Web Services Addressing SOAP Binding	09 May 2006	Recommendation
Web Architecture	15 Dec 2004	Draft /Proposal

Structure of WSDL 1.1 document

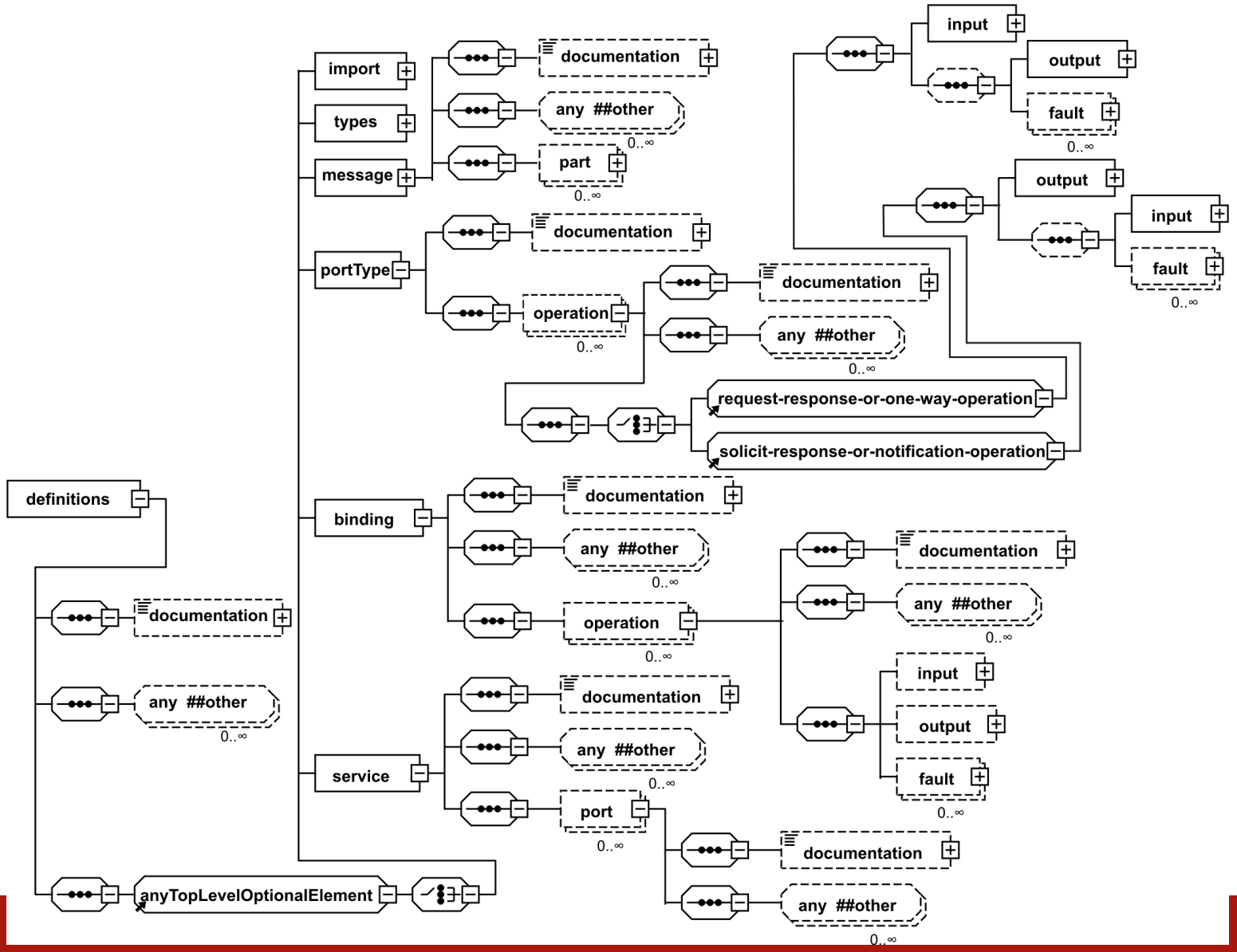
- The abstract section includes: port types, messages and types constructs.
  - Port types are abstract collections of supported operations.
  - Operations refer to messages that are abstract descriptions of the data being exchanged.
  - Operations and messages are bound to a concrete network protocol and message format.
- The concrete section includes: binding that defines the concrete protocol and data format specifications for a particular port type.
  - Port definition associates a network address with a reusable binding.
  - A collection of ports defines a service.

Structure of WSDL 2.0 document

- The abstract section includes: interfaces and types constructs.
  - An interface collects together supported operations.
  - An operation associates a message exchange pattern with one or more messages.
  - Messages are described using a type system (usually XML Schema) for defining bodies of inputs, outputs and faults.
- The concrete section includes: binding that specifies transport and wire format details for one or more interfaces.
  - An endpoint associates a network address with a binding.
  - A group of endpoints that implement a common interface defines a service.


WSDL 1.1
WSDL 2.0
definitions
description
types
Element decl.
Type def.
types
Element decl.
Type def.
message
portType
operation
input
output
fault
interface
operation
input
output
infault
outfault
fault
MEP
Operation style
Abstract section
Concrete section
binding
service
port
binding
service
endpoint

WSDL 1.1


definitions
documentation
any ##other
0..∞
anyTopLevelOptionalElement
import
types
message
documentation
any ##other
part
0..∞
portType
documentation
operation
0..∞
documentation
any ##other
0..∞
request-response-or-one-way-operation
solicit-response-or-notification-operation
input
output
fault
0..∞
output
input
fault
0..∞
binding
documentation
any ##other
0..∞
operation
0..∞
documentation
any ##other
0..∞
input
output
fault
0..∞
service
documentation
any ##other
0..∞
port
0..∞
documentation
any ##other
0..∞

<definitions>

- This is a root element of WSDL document. It works as a container for several fragments (nested elements) that forms together a full service description.
- It provides namespace declarations valid for its content along with XML namespace prefixes.

<documentation>

- Element documentation is used to provide a human readable description. This element can assists (be nested in) any other element appearing in a WSDL document.

<import>

- This elements works like #include preprocessor directive in C programming language. It allows splitting the description into independent documents and integrating them in one, main document as necessary. This improves the modularity and legibility definition of the service. The import has two attributes: namespace and location.

<types>

- This element is a container for data types definitions used in <message> elements. It contains zero or more sub-elements <schema>, which must adhere to the rules for XML Schema documents. The declared types can be complexType or simpleType.

<message>

- This element defines the format of messages exchanged between a client and a web service. It may represent a query, a response or a signal on error. It refers to the data types defined in <types>. The data contained in <message> are abstract. A message consists of one or more sub-elements <part>

<operation>

- is an abstract definition of an operation supported by a Web service
- there can be several input, output and fault messages defined declared using nested <input>, <output> and <fault> elements. These elements refer to <message> elements defined in the same WSDL document or imported from external documents.
- the order of messages should follows so called Message Exchange Patterns (MEP).

<binding>

- This elements is used to specify a concrete protocol binding and data encoding for a given <portType> (i.e. it provides binding to HTTP, SOAP MIME or, possibly, custom protocols).
- Since in the WSDL document <operation> elements are already defined, the element <binding> maps the abstract definitions of operations, their input and output messages, to the appropriate protocol used by a web service.

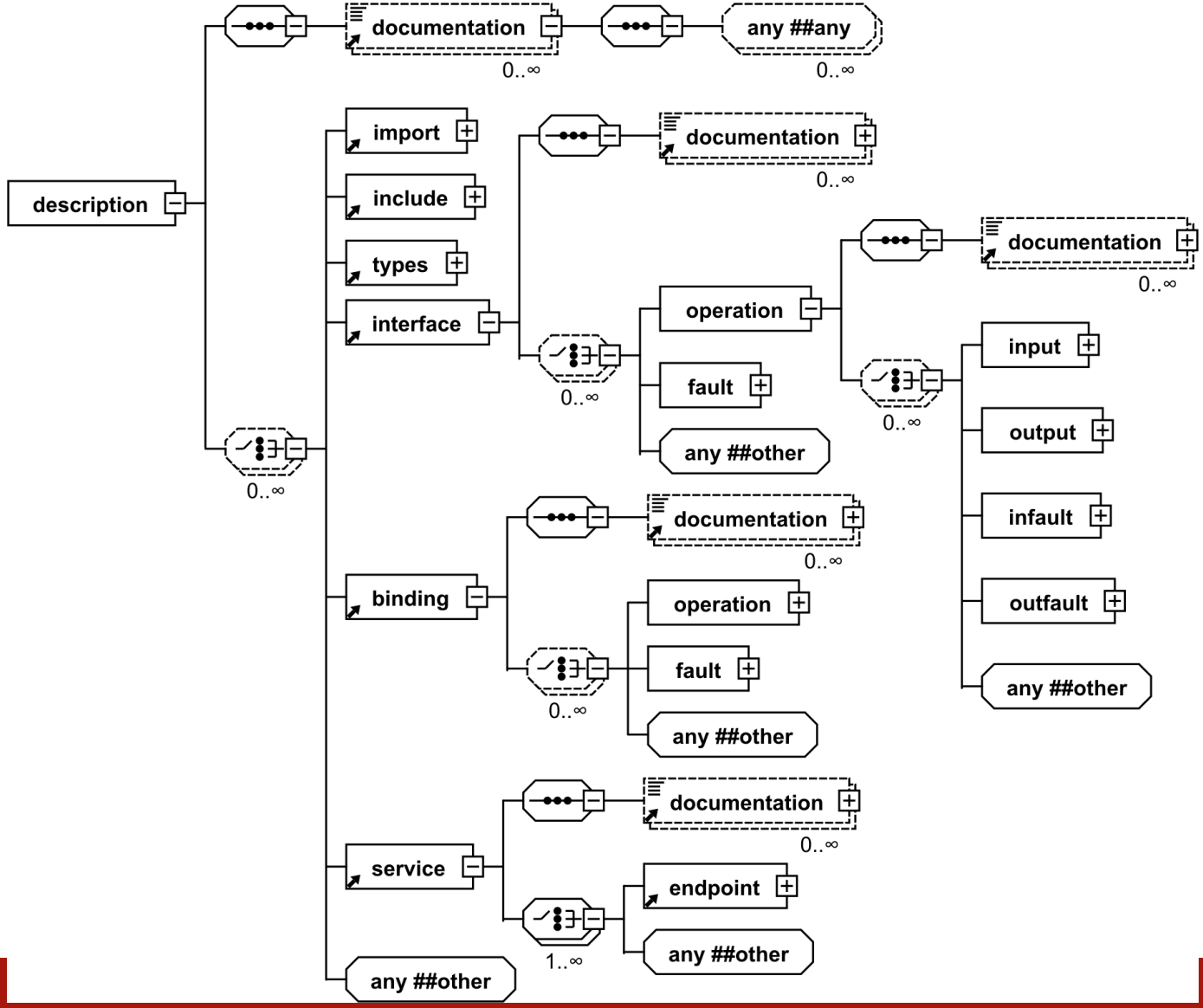
<portType>

- This element specifies a set of operations supported by the service endpoint (it provides a unique identifier to a group of operations supported by a single endpoint). Each <operation> is defined individually.

<service>

- This element appears typically at the end of the WSDL document. It defines a concrete web service endpoint with URL to the service location (there is no other occurrences of such URL before service element). <service> element groups one or more <port> elements. A single <port> element represents an endpoint (access point) to a web service.

WSDL 2.0


description
documentation
0..∞
any ##any
0..∞
import
include
types
interface
0..∞
documentation
0..∞
operation
fault
any ##other
documentation
0..∞
input
output
infault
outfault
any ##other
0..∞
binding
documentation
0..∞
operation
fault
any ##other
0..∞
service
documentation
0..∞
endpoint
any ##other
1..∞
any ##other

<description>

- The <description> is a main element of a WSDL document. The content of this element should conform to the following pattern:

```
<description
        targetNamespace="xs:anyURI" >
  <documentation />*
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service
    /> ]*
</description>
```

<documentation>

- A human readable description in the WSDL 2.0 documents is provided within optional <documentation> elements. These elements can appear in all elements included in a <description>. The <documentation> syntax is following:

```
<documentation>
   [extension elements]*
</documentation>
```

<include> and <import>

- Element <include> allows to include components defined in other WSDL documents in the current interface definition. Element <include> has a mandatory location attribute which specifies the location of the external, included WSDL documents. The target namespace of attached WSDL descriptions must match the target namespace of the base document.
- Element <import> has a similar meaning as <include>. The difference is that the imported WSDL document can have different target namespace then the base document. This element has a mandatory namespace attribute for an imported element and an optional location attribute. Standard content of this element is as follows:

<types>

- Element <types> serves as a place for definitions of data types used by exchanged messages.
- XML Schema is preferred as typing language, although it is possible to use also the DTD or RELAX NG.
- The use of custom schemas relies on importing them (with the use of <xs:import>) or embedding them within <types> element of the WSDL document (using <xs:schema>).
- The elements from imported or included schemas can be referenced by using their QName.

<interface>

- Element <interface> contains <operation> elements, which group definitions of sequences of messages sent to and from the service.
- It may contain <fault> elements with errors descriptions.
- It has mandatory attribute name (the value of which is a name of the interface), and two optional attributes: extends (which lists the interfaces that the interface extends) and styleDefault (which contains the default style used to create element declarations for all interface operations).

<fault>

- This element serves as an abstract description of a fault that MAY occur during invocation of an operation of an interface.
- The two attributes, name and element, are used, respectively, to declare the name of the fault and indicate the contents of the fault message.

<operation>

- This element describes interaction with a service by listing messages exchanged between the service and its user or users
- It may contain such elements as: <input>, <output>, <infault>, and <outfault>.
- The list of attributes of the <operation> consists of the following items: style, safe, pattern (all three are optional), and name (required).

<input> and <output>

- These elements are message containers used in normal communication. Message exchange pattern includes rules on how to generate messages with information about the error.
- Elements <infault> and <outfault> are used as containers for messages with errors. Signaling an error might terminate message exchange pattern. The value of their mandatory messageLabel attribute indicates the role of messages they carry.

<binding>

- This element contains binding declarations necessary to access the service. These declarations describe a message format and transmission protocol for communication with an endpoint (including encoding of input and output parameters implemented in the service).
- The relevant declarations are described in the mandatory element <operation>.
- once defined a <binding> can be used many times in the definitions of various interfaces.

<service>

- Element <service> contains a set of <endpoint> elements, determining the places where the service has been implemented. An interface attribute of the service contains the name of the interface, whose endpoints are just being described.

<endpoint>

- Element <endpoint> is used to define an endpoint.
- Two of its attributes - name (name of the endpoint) and binding (representing binding) are required.
- The third attribute, address (address of the service that implements the interface) is optional.

Extensible Markup Language (*XML*)

- metalanguage used to represent and manipulate data elements
- can be used to encode data transmitted from one place to another
- using XML Schema one can expresses strict hierarchical relationships immediately, without a conceptual data model

Extensible Markup Language (*XML*)

- metalanguage used to represent and manipulate data elements
- hundreds XML-based languages have been developed:
  - RDF/XML, Relax NG, XML Schema, SOAP, SVG … (https://en.wikipedia.org/wiki/List_of_XML_markup_languages)
- commonly used to encode data transmitted from one place
- associated Internet media type, formerly known as **MIME type** (*Multipurpose Internet Mail Extensions*)
  - for xml documents without any special semantics
    - `application/xml`
    - `text/xml`
  - for xml documents written in XML-based languages
    - media types ending in `+xml`, for example `image/svg+xml` for SVG
- standards
  - https://www.w3.org/TR/?tag=xml&status=rec

XML syntax

- Processor and application
  - *processor* analyses the markup and passes structured information to an *application*
- Tag
  - begins with < and ends with > and comes in three flavours:
    - start-tag, such as `<element>`,
    - end-tag, such as `</element>`
    - empty-element tag, such as `<element />`
- Attribute
  - consisting of a name–value pair that exists within a start-tag or empty-element tag
    - `attributename="attribute-value"`
- Element
  - a logical document component
  - begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag
  - all characters between the start-tag and end-tag, if any, are the element's *content*,
  - the content may contain markup, including other elements, which are called *child elements*.

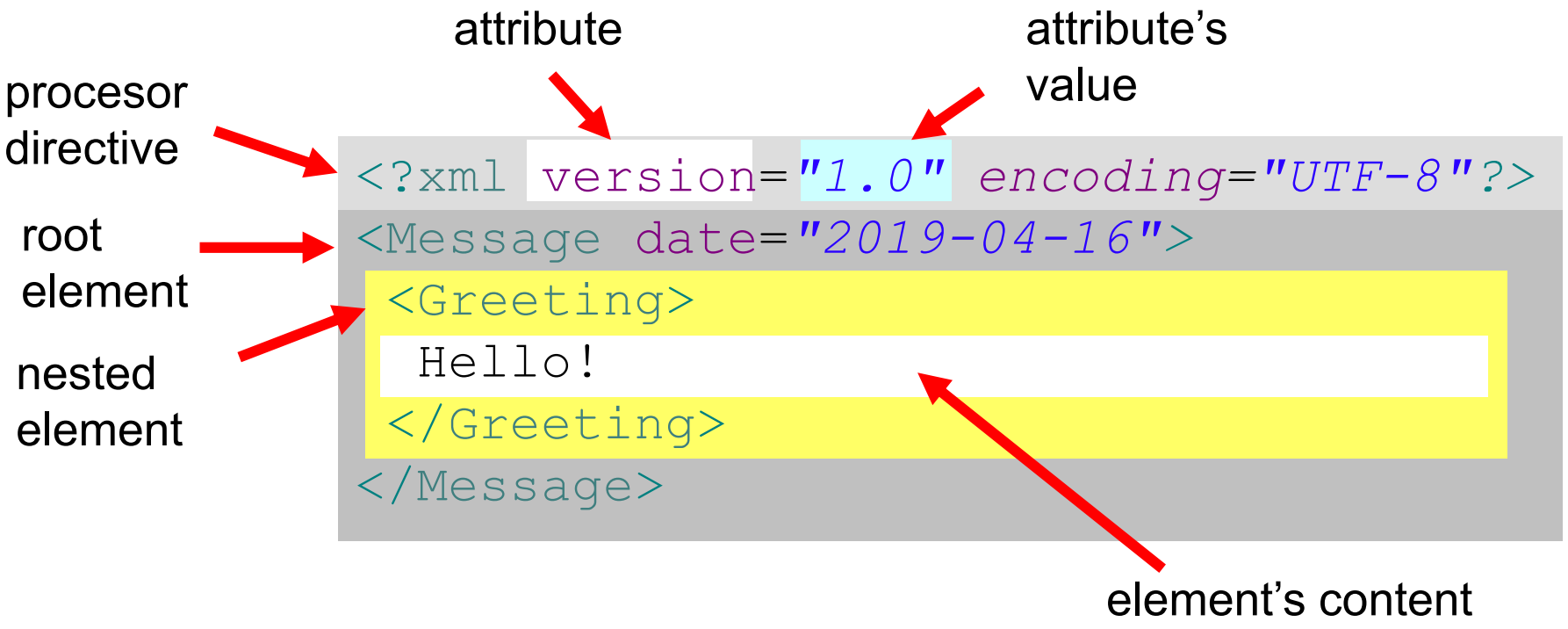
XML document structure

- Non-empty element

```
<tag attributename="attribute-value">content</tag>
```

- Empty element

```
<tag attributename="attribute-value"/>
```



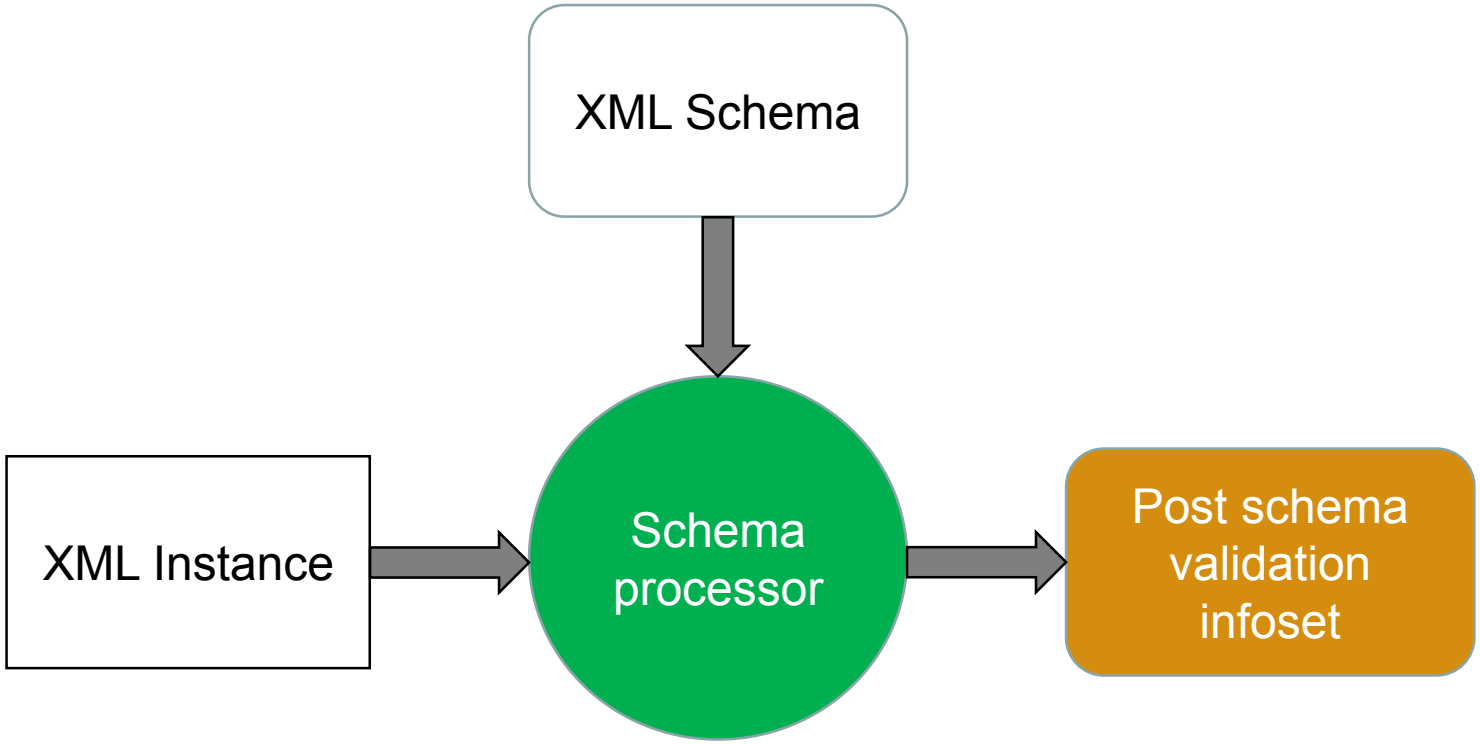
Well-formed and Valid documents

- Well-formed xml document
  - there is only one root element
  - all other elements are cleanly nested (overlapping elements are not allowed)
  - non-empty element must have both: start and end tags

    `<element> </element>`
  - all attribute values must be enclosed in quotation mark and assigned with the use of = operator

- Valid xml document
  - well formed
  - conforms to the predefined schema

XML Validation



DTD - document type definition

- part of the XML language specification (XML 1.0 & 1.1 spec)
- defines the structure and the legal elements and attributes of an XML document
- can be embedded inside an XML document or can be referenced
- is used by applications to verify that XML data is valid

DTD primitive types

- CDATA
  - a string type with no restrictions placed on it, it can contain any textual data as long as its suitably escaped
- ENTITY
  - has to start with a letter or ':' or '_' and the rest of the characters must be numbers, letters ':', '_', '-', or '.' but no space
  - the value of the ENTITY attribute also has to match an ENTITY that has been declared elsewhere within the DTD
- ENTITIES
  - a list of values like in ENTITY separated by a single space
  - each value within the entities attribute has to match an ENTITY that has been declared elsewhere within the DTD
- DTD ID
  - must conform to the naming rules like ENTITY
  - must be unique across all the ID values declared within the XML Document
  - an element may only have a single ID value declared against it
  - an attribute declared of type ID must be defined as either #IMPLIED or #REQUIRED
- IDREF
  - must conform to the naming rules like ENTITY
  - IDREF attribute has to match an ID value defined elsewhere within the XML Document.
- IDREFS
  - a list of values that conform to the naming rules like ENTITY separated by a single space
  - each value within the IDREFS attribute must match an ID value that has been defined elsewhere within the XML Document.
- NMTOKEN
  - can contain numbers letters, and ':', '_', '-', or '.' it can not contain spaces or whitespace
- NMTOKENS
  - it must contain one or more NMTOKEN values separated by a single space
- PCDATA (parse-able text data)
  - an element can contain #PCDATA, but not attribute
  - represents mixed content: elements may contain character data, optionally interspersed with child elements

https://www.liquid-technologies.com/DTD/Datatypes/index.aspx

DTD Syntax Glossary

DTD Expression	Description
CDATA	Data Type : Character Data
ID	Data Type : Unique Identifier
IDREF	Data Type : Reference to a Unique Identifier
IDREFS	Data Type : Reference to a 0-n Unique Identifiers
ENTITY	Data Type : Reference to an ENTITY Declaration
ENTITIES	Data Type : Reference to an 0-n ENTITY Declarations
NMTOKEN	Data Type : Named Token
NMTOKENS	Data Type : 0-n Named Tokens
?	Element cardinality : Optional (0-1)
*	Element cardinality : Zero to Many (0-n)
+	Element cardinality : One to Many (1-n)
,	Sequence : separates items within a sequence
|	Choice : separates items within a choice
-->	Ends a comment
<!--	Starts a comment
ANY	Declares an attribute on an Element

https://www.liquid-technologies.com/DTD/Datatypes/index.aspx

DTD Syntax Glossary

DTD Expression	Description
ATTLIST	Declares an attribute on an Element
EMPTY	Declares an attribute on an Element
DOCTYPE	Starts a DTD declaration
ELEMENT	Defines an XML Element
ENTITY	Defines an ENTITY, which can be used as a replacement token
NOTATION	
#PCDATA	Allows mixed content to be defined
#REQUIRED	Attribute property : Attribute is required
#FIXED	Attribute property : Attribute must have a specific value
#IMPLIED	Attribute property : Attribute is optional (0-1)
SYSTEM	ENTITY property : The entity value is to be loaded from a uri
PUBLIC	ENTITY property : The entity value is referred to by a public identifier, or can be loaded from a uri

https://www.liquid-technologies.com/DTD/Datatypes/index.aspx

DTD based validation

XML file `message.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Message SYSTEM "message.dtd">
<Message date="2019-04-16">
    <Greeting>Hello!</Greeting>
</Message>
```

DTD file `message.dtd`

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT Message (Greeting)>
<!ELEMENT Greeting (#PCDATA)>
<!ATTLIST Message date CDATA #REQUIRED>
```

XML Schema based validation

XML file `message.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MySchema.xsd" date="2019-04-16">
    <Greeting>Hello!</Greeting>
</Message>
```

XML Schema file `message.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
    <xs:complexType name="MessageType">
    <xs:sequence>
    <xs:element name="Greeting" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="Date" type="xs:date" />
    </xs:complexType>

    <xs:element name="Message" type="MessageType" />
</xs:schema>
```

XML Schema based validation

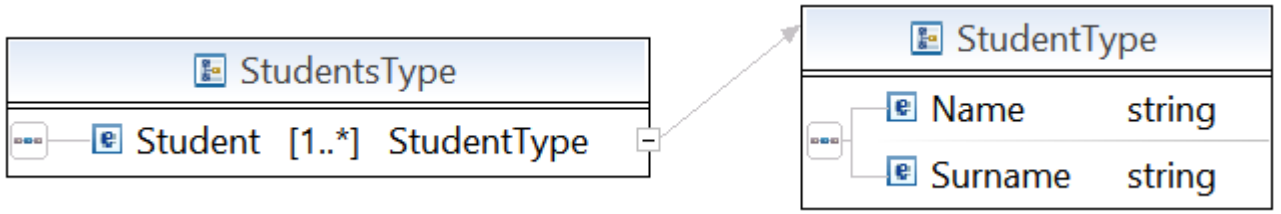
XML Schema file `student.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/MySchema"
xmlns:tns="http://www.example.org/MySchema" elementFormDefault="qualified">

    <complexType name="StudentsType">
    <sequence>
    <element name="Student" type="tns:StudentType" maxOccurs="unbounded"
minOccurs="1"></element>
    </sequence>
    </complexType>

    <complexType name="StudentType">
    <sequence>
    <element name="Name" type="string"></element>
    <element name="Surname" type="string"></element>
    </sequence>
    </complexType>

    <element name="Students" type="tns:StudentsType"></element>
</schema>
```

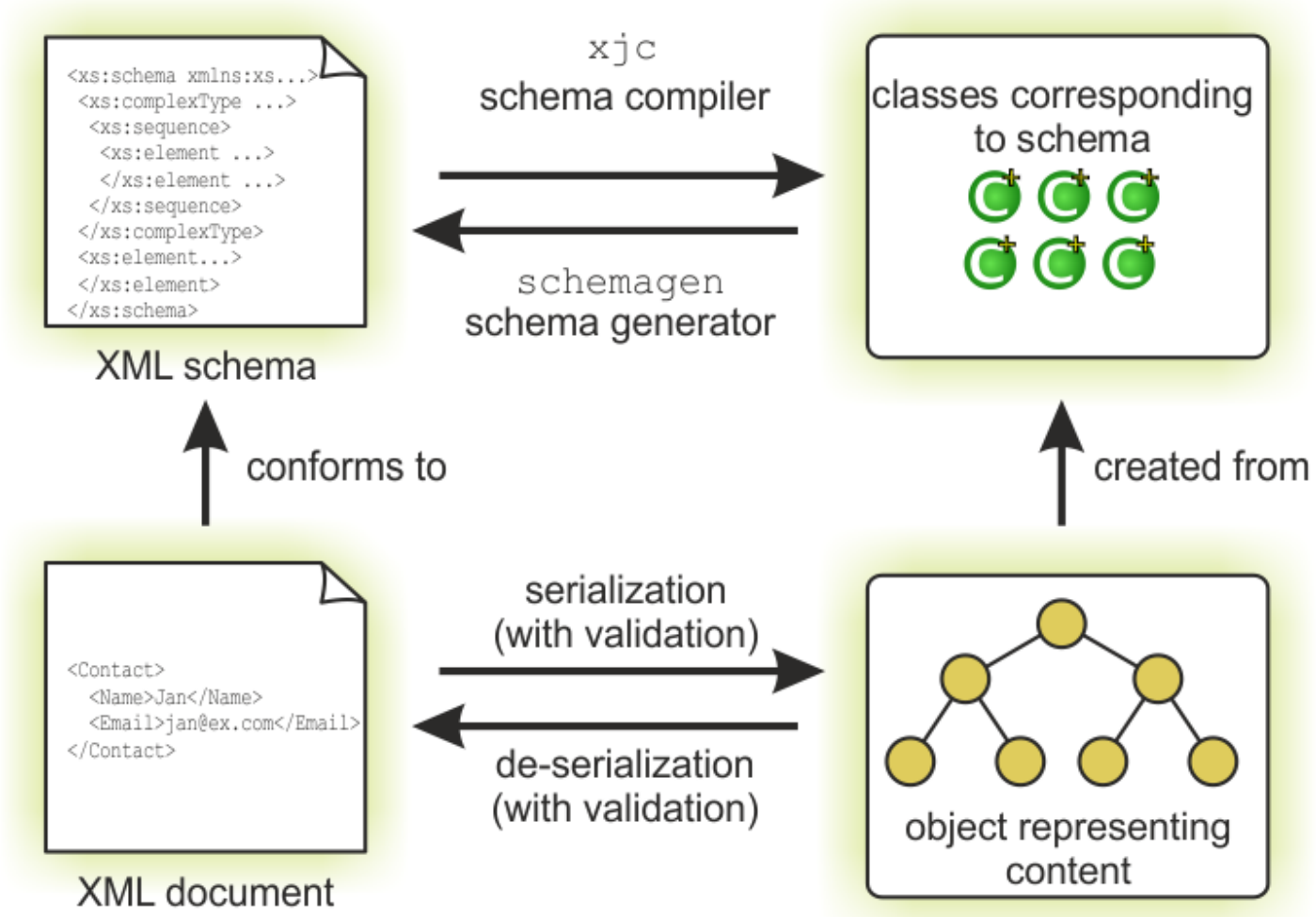


XML Schema based validation

XML file `student.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Students xmlns:tns="http://www.example.org/MySchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.org/MySchema
student.xsd ">
  <tns:Student>
    <tns:Name>Adam</tns:Name>
    <tns:Surname>Nowak</tns:Surname>
  </tns:Student>
</tns:Students>
```

Using JAXB – round trip

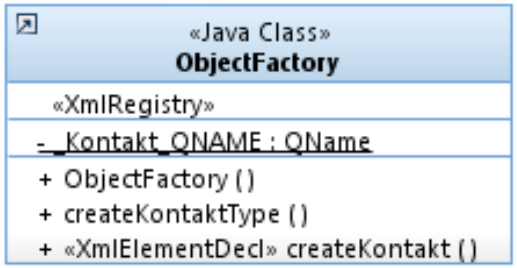


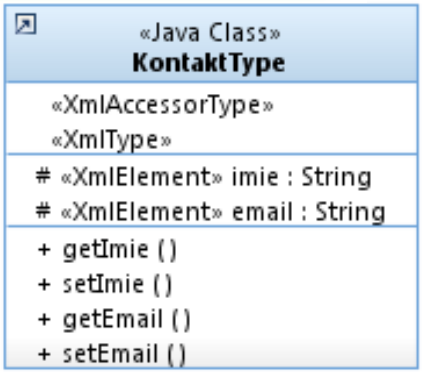
Nice example:
http://javajee.com/lab-jaxb-2-marshaling-and-unmarshaling-with-xsd-to-java-binding

Using JAXB

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs="http://www.w3.org/2001/XMLS
   chema">
 <xs:complexType name="KontaktType">
  <xs:sequence>
   <xs:element name="Imie"
   type="xs:string"></xs:element>
   <xs:element name="Email"
   type="xs:string"></xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="Kontakt"
   type="KontaktType"></xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
 <Kontakt>
  <Imie>Jan</Imie>
  <Email>jan@ex.com</Email>
 </Kontakt>
```






```
> xjc dane.xsd
```

```
JAXBContext jc = JAXBContext.newInstance("generated" );
        Unmarshaller u = jc.createUnmarshaller();
        FileInputStream f = new FileInputStream("mojeDane.xml");
        JAXBElement<generated.KontaktType> k =
            (JAXBElement<generated.KontaktType>) u.unmarshal(f);
        System.out.println(k.getValue().getImie());
        System.out.println(k.getEmail ().getEmail());
```

Web services

- Creating bottom-up and top-down Web services
  - https://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/BottomUpWebService/BottomUpWebService.html
- Eclipse Help
  - https://help.eclipse.org/2019-03/index.jsp
- JAX-WS Tools User Guide
  - https://help.eclipse.org/2019-03/index.jsp
  - **Testing and validating Web services**
    - https://help.eclipse.org/2019-03/index.jsp?topic=%2Forg.eclipse.wst.wsi.ui.doc.user%2Ftasks%2Ftmonitor.html
- **Introduction to Apache CXF**
  - https://www.baeldung.com/introduction-to-apache-cxf