Distributed systems – Techs
7. WS standards: XML, WSDL, SOAP, UDDI

XML

Why XML?

- Web pages – need a human to understand what they mean.
- What if Web info is available in a form that could be easily used by other pros?
- Early days of the Web: *screen scraping* – HTML is parsed and its meaning is inferred based on assumptions about page layout, table headings, etc
  - lost cause because Web designers change page layout frequently to keep their sites interesting.
- XML – eXtensible Markup Language (W3C, 1998) - initially a "better HTML"
  - Soon became apparent that the real spot for XML was as a data interchange format.
- Make info available to other progs on the Web: publish it in XML format.
  - Need to define an XML vocabulary that describes your application data or use an industry standard vocabulary if a suitable one exists.
  - The predominant activity following the publication of the XML specification was the definition of standard vocabularies such as
    - Mathematical Markup Language (MathML),
    - Chemical Markup Language (CML)
    - Meat and Poultry Markup Language (mpXML)
    - …
- XML : textual and architecturally neutral so there was no confusion about low-level details such as the order of bytes in an integer.

Overview of XML

- Allows the structured representation of arbitrary data.
- XML files are simple text files that can be edited with any editor.
- XML is called a markup language because the data is “marked up” through *tags*.
- Example: an XML specification

```
<Person>
    <FirstName>Mickey</FirstName>
    <LastName>Mouse</LastName>
    <Age>75</Age>
</Person>
```

  - Person, FirstName, LastName, and Age are tags.
  - A *start tag* is surrounded by “<”and“>”while an *end tag* is surrounded by “</”and “>”.
  - Identifiers Person, FirstName are application specific, not part of the XML standard.
  - Between the start tag and the end tag is the *content* of the tag.
  - The combination of start and end tags and the content is referred to as an *element*.
  - Tags can be the content of other tags; tag Age belongs to the content of tag Person.
  - Tags have to be strictly nested, which results in a hierarchical or tree-like representation of the data.

Overview of XML

- Tags can have one or more attributes – example (“type definition”):

```
<struct name="Person">
    <member type="string" name="first_name"/>
    <member type="string" name="last_name"/>
    <member type="int" name="age"/>
</struct>
```

- Name and type are called *attributes , e.g.* name is an attribute of tag struct.
- The value of an attribute is written between the double quotes, e.g. person is the value of attribute name.
- The values of the attributes belong to the content of a tag,
  - there are no absolute rules whether data should be placed as the content of a tag or as a value of an attribute of that tag.
- If there is no content for a tag, the tag can be surrounded by “<”and“/>” instead of an explicit end tag.
- XML can be used to describe both the types and instances of data (first example) to be handled by a middleware.

XML Parsers

- To read an XML document, an application uses a *parser* to get the data contained in the document.
- A parser usually consists of a large API that allows the programmer to choose which elements to look at in the document.
- Microsoft's *.NET* architecture provides a developer with several classes for accessing the data in a document,
- *Apache* group develops a parser called *Xerces* that works cross platform.
- With Web Services, an application passes an XML document across the Internet with different transport protocols.
  - Therefore, either a client or sever side program must parse the XML to get to the data within the document.

Processing instruction & Root element

- The first part of any XML document is the *Processing Instruction* (PI).
  - This tells the parser that the data is in an XML document and the version of XML used (at this point it's always 1).
  - The start of the document now looks like the following:

    <?xml version="1.0" ?>

- To begin describing data, a root element must be present.
  - This is the outermost element in the document.
  - An element is simply a tag that looks much like an HTML tag, but in the case of XML the programmer chooses the name of the tag.
  - Example: BOOK is the root element.

    <?xml version="1.0" ?> <BOOK> </BOOK>
  - An XML document must have only one root element
  - Usually, the root element begins the definition of a *SOAP* document or a WSDL file.

Attributes

- Additional information added to an element is an *attribute*
- Example:

```
<?xml version="1.0" ?>    <BOOK TITLE="Distributed Systems">
    </BOOK>
```

- Attributes always appear as part of the opening element and can be in any element in the document
- Attributes often define namespaces or locations, such as the next *SOAP* node, for the XML document.
- Attribute centric document: the information all resides within attributes

```
<?xml version="1.0" ?>
<BOOK TITLE="Distributed Systems"
  PAGECOUNT="400"
  AUTHOR="Ion Ionescu"
  PUBLISHER="New House Press"/>.
```

Namespaces

- To enable using the same name with different meanings in different contexts, XML schemas may define a namespace.
- A *namespace* = a set of unique names that are defined for a particular context and that conform to specific rules
  - Namespaces ensure that the element names used in your XML document are unique.
  - The namespace definition occurs in the root element (the outermost element) and utilizes a URL as a unique identifier.
- Example:

  <BOOK XMLNS:WEBSERVICES="www.newhouse.com/XML"></BOOK>
  - Then all the child elements of BOOK begin with the namespace.

```
<?xml version="1.0" ?>    <BOOK
    XMLNS:WEBSERVICES="www.newhouse.com/XML">
    <WEBSERVICES:TITLE>Distributed Systems</WEBSERVICES:TITLE>
    <WEBSERVICES:PAGECOUNT>400</WEBSERVICES:PAGECOUNT>
     <WEBSERVICES:AUTHOR>Ion Ionescu</WEBSERVICES:AUTHOR>
     <WEBSERVICES:PUBLISHER>New House Press</ WEBSERVICES:PUBLISHER>
 </BOOK>
```

- Namespaces in WS: utilize them usually as a way to represent various elements that are vendor dependent or to support primitive types from schemas.

XML namespaces

- An XML namespace defines a collection of names and is identified by a URI reference.
- Example: xmlns="http://simple.example.com/CInfoXmlDoc".
- Names in the namespace can be used as element types or attributes in an XML document.
- The combination of URI and element type or attribute name comprises a unique universal name that avoids collisions.
- Example: a namespace that defines the ContactInformation document's element types, such as Name and Address.
  - these element types are unique within the contact information context.
  - if the document included another namespace context, such as BankInformation that defined its own Name and Address element types, these two namespaces would be separate and distinct.
  - a Name and Address used in the context of BankInformation would not conflict with a name and address used in the context of ContactInformation.

Well-formed and valid XML

- Means that the document meets all the rules and contains all the information specified in either a *Document Type Definition* (DTD) or in an *XML schema definition* (XSD).
  - Both act as a packing slip for XML documents, specifying which data needs to be present in the document.
  - Validating a document against a schema or a DTD is a costly process and, thus, probably only occurs during the development of *SOAP* software.
- A well-formed XML document follows the rules set forth by the W3C:
  - there must be one or more elements,
  - there can only be one root element that is not overlapped by any other element,
  - every start tag must have an end tag unless it's an empty element.
- Example 1:

  <?xml version="1.0" ?> <BOOK TITLE="Distributed Systems">

  Is not well formed (missing / at the end)
- Example 2: the root element gets overlapped by another tag

```
<?xml version="1.0" ?>
  <BOOK TITLE="Distributed Systems ">
      <AUTHOR> Ion Ionescu </AUTHOR>
      <BOOKDATA>  <PAGECOUNT>400</PAGECOUNT>
     <PUBLISHER> New House</PUBLISHER>   </BOOK>   </BOOKDATA>
```

DTD and XSD

- A DTD or XSD describes the structure of an XML document.
  - information on the tags the corresponding XML document can have, the order of those tags, and so forth.
  - Validating an XML document ensures that the document follows the structure defined in its DTD or XSD and that it has no invalid XML tags.
  - Systems exchanging XML documents for some purpose can agree on a single DTD or XSD and validate all XML documents received for that purpose against the agreed-upon DTD/XSD before processing the document.
- DTDs are a hold over from the *Serialized General Markup Language* (SGML) standard that the publishing industry created to publish books.
  - Allow a user to specify entity references that gave the ability to substitute values in and out of XML documents.
- Disadvantages of DTD:
  - Do not provide things needed for common programming such as types or order.
  - DTDs provide no real facility to express data types or complex structural relationships.
- XSD standardize the format definitions of XML documents.

Example of DTD

A simple DTD for previous example:

- need to recognize that BOOK is the root element and TITLE, PAGECOUNT, AUTHOR, and PUBLISHER are all children of BOOK.
- because BOOK is a root element, it cannot be optional but all the other elements can be.
- need the DTD to recognize that TITLE is an attribute of BOOK.

```
<?xml version="1.0" ?>
<!-- This is a comment -->
<!-- The following code is the DTD -->
<!-- The PI and the DTD are the prolog of the document -->
<!DOCTYPE BOOK [
<!ELEMENT BOOK (PAGECOUNT?,AUTHOR+,PUBLISHER+)>
<!ATTLIST BOOK TITLE CDATA #REQUIRED>
<!ELEMENT PAGECOUNT (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
 <!ELEMENT PUBLISHER (#PCDATA)>    ]>
<BOOK TITLE="Distributed Systems">
<PAGECOUNT>400</PAGECOUNT>
<AUTHOR>Ion Ionescu</AUTHOR>
<PUBLISHER>New House Press</PUBLISHER>
</BOOK> The DTD at the beginning of the document is clearly not XML.
```

Specifying quantities is done with the symbols +, *, and ?.
The + means 1 or more of the element whereas the * means 0 or more.
The ? means the element is optional.

Another example: XML and DTD

```
<?xml version="1.0" encoding="ISO-8859-1"
    standalone="yes"?>
<ContactInformation>
  <Name>Ion Ionescu </Name>
   <Address>
      <Street>B-dul V.Parvan 4</Street>
      <City>Timisoara</City>
      <State>Timis</State>
      <Country>RO</Country>
   </Address>
   <Phone>0256666333</Phone>
   <EMail>ion_ionescu@yahoo.com</EMail>
 </ContactInformation>
```

```
<!ELEMENT ContactInformation
  (Name, Address, Phone,EMail)>
<!ELEMENT Name(#PCDATA)>
<!ELEMENT Address (Street, City, State,
    Country)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Phone (#PCDATA)>
<!ELEMENT EMail (#PCDATA)>
```

XML and XSD

```
<?xml version="1.0" encoding="ISO-8859-1"
      standalone="yes"?>
<ContactInformation
      xmlns="http://simple.example.com/CInfoXmlDoc"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance"       xsi:schemaLocation=
      "http://simple.example.com/CInfoXmlDoc
      file:./CInfoXmlDoc.xsd">
<Name> Ion Ionescu </Name>
 <Address>
<Street>B-dul Vasile Parvan 4</Street>
<City>Timisoara</City>
<State>Timis</State>
<Country>RO</Country>
</Address>
<HomePhone>0256666333</Phone>
<EMail> ion_ionescu@yahoo.com </EMail>
</ContactInformation>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://simple.example.com/CInfoXmlDoc"
      xmlns=" http://simple.example.com/CInfoXmlDoc"
      elementFormDefault="qualified">
<xsd:element name="ContactInformation">
 <xsd:complexType>
  <xsd:sequence>
     <xsd:element name="Name" type="xsd:string" />
     <xsd:element name="Address">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Street"  type="xsd:string" />
             <xsd:element name="City"    type="xsd:string" />
             <xsd:element name="State"   type="xsd:string" />
             <xsd:element name="Country" type="xsd:string" />
           </xsd:sequence>
          </xsd:complexType>
       </xsd:element>
       <xsd:element name="HomePhone" type="xsd:string" />
       <xsd:element name="EMail" type="xsd:string" />
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Schema generated by Visual Studio .NET

```
<?xml version="1.0" ?>
 <xs:schema id="NewDataSet"
        targetNamespace="http://www.newhouse.com/~vs1C0.xsd"
        xmlns:mstns="http://www.newhouse.com/~vs1C0.xsd"
        xmlns="http://www.newhouse.com/~vs1C0.xsd"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
        attributeFormDefault="qualified"
        elementFormDefault="qualified">
<xs:element name="BOOK">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="PAGECOUNT" type="xs:string"
        minOccurs="0"          maxOccurs="3"  msdata:Ordinal="0" />
       <xs:element name="AUTHOR" type="xs:string" minOccurs="0"
        msdata:Ordinal="1" />
       <xs:element name="PUBLISHER" type="xs:string" minOccurs="0"
        msdata:Ordinal="2" />
    </xs:sequence>
   <xs:attribute name="TITLE" form="unqualified" type="xs:string"/>
   </xs:complexType>
  </xs:element>
 <xs:element name="NewDataSet" msdata:IsDataSet="true"
        msdata:EnforceConstraints="False">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
     <xs:element ref="BOOK" />
    </xs:choice>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

- *Visual Studio.NET* generate schemas automatically based on a given XML document
- namespaces defined and many of them deal specifically with things *Visual Studio* needs
- The first xs:element defines the requirement for PAGECOUNT in the XML document:
  - it is a string according to the type attribute,
  - minOccurs set to 0 indicates that PAGECOUNT is not required,
  - maxOccurs means that PAGECOUNT can appear a maximum of three times.
- xs:sequence tag allows to determine the order of the elements in the schema.

Advantages of XML

- designed exactly for data exchange purposes and has demonstrated its strength over time.
- simple, flexible, text-based markup language.
- standard accepted throughout the industry, enables service providers and requestors to communicate with each other in a common language
- not dependent on a proprietary platform or technology, and messages in XML can be communicated over the Internet using standard Internet protocols such as HTTP.
- product of the World Wide Web Consortium (W3C) body => changes to it will be supported by all leading players.
- This ensures that as XML evolves, Web services can also evolve without backward compatibility concerns.
- Main: Structured, Portable, Extensible, Text format

WSDL

Service contracts

- Every service has a well-defined, formal interface called its service contract that
  - clearly defines what the service does
  - clearly separates the service's externally accessible interface from the service's technical implementation.
- Elements of Service Contract:
  - Service names: Human-friendly name (plus aliases), unique machine-readable name.
  - Version number: Supports the service lifecycle.
  - Pre-conditions: Conditions that must be satisfied prior to using the service for example, an operation may not be accessible between midnight and 2 am.
  - Service classification: Notes and keywords that identify the business domain(s) that the service supports "yellow page" entries for the service.
- The service contract can be
  - explicit defined using WSDL, XML Schema, and the WS-Policy framework, or
  - implicitly defined based on the input messages the service accepts, the output messages it responds with, and the business activities that it implements
- For Web services-based SOA implementations,
  - WSDL is used to define key elements of the service contracts,
  - other elements that cannot be expressed in WSDL are defined using the WS-Policy framework or documented in a document or spreadsheet.

Web Services Description Language - evolution

- defines a standard way for specifying the details of a Web service.
- is a general-purpose XML schema that can be used to specify details of Web service interfaces, bindings, and other deployment details.
- describes
  - what public methods are available and
  - where the service is located.
- WSDL 1.1 was made available in 2001
  - In 2002, a Working Draft of WSDL 1.2 was released.
  - WSDL is pretty much the universally accepted protocol for describing Web services.
- WSDL 2.0 is a W3C Recommendation issued in 2007.
  - WSDL 1.1 and 2.0 are conceptually very similar.
  - WSDL 2.0 benefited from a long and careful review process.
    - WSs that support Web architectural principles such as REST are the first adopters of WSDL 2.0.

Structure of the WSDL Document

Six major elements are included in a WSDL document:

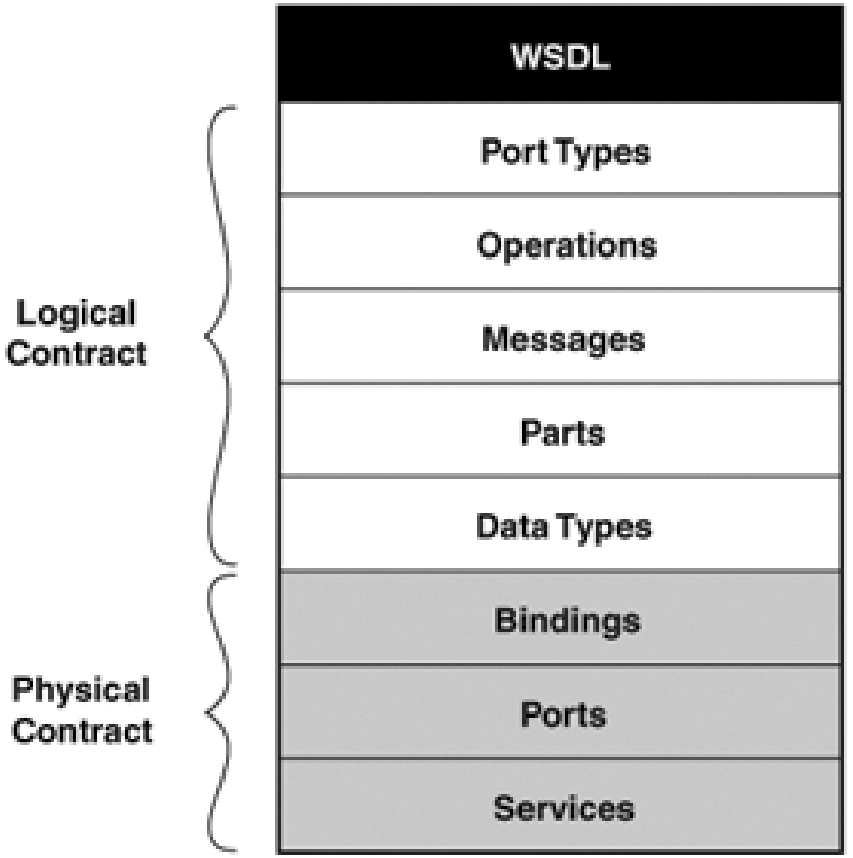
- The data types that the WS uses
- The messages that the WS uses
- The operations that the WS performs
- The communication protocols that the WS uses
- The individual binding addresses
- The aggregate of a set of related ports

- Port describes the *what* of a WS,
- Binding describes the *how*,
- Service describes the *where*.

WSDL Structure

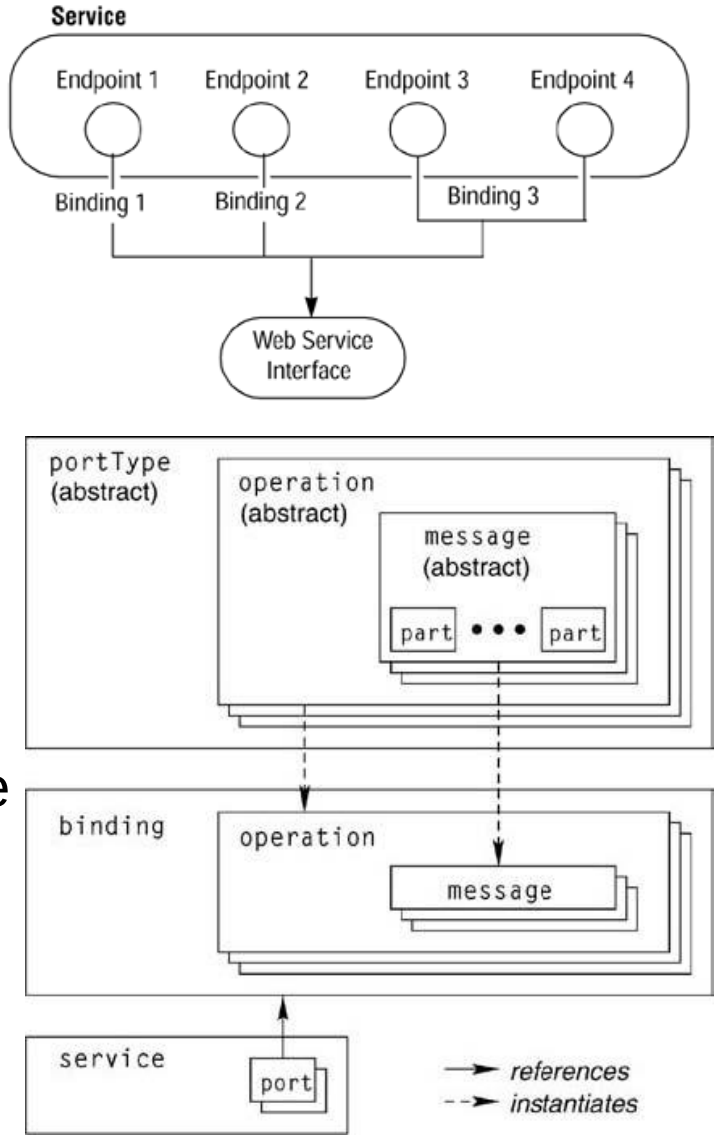
1. The logical contract defines the public interface that is independent of transports, on-the-wire data formats, and programming languages.

2. The physical contract defines bindings to transports and wire-level data formats, and multiple physical contracts can be defined for each logical contract.



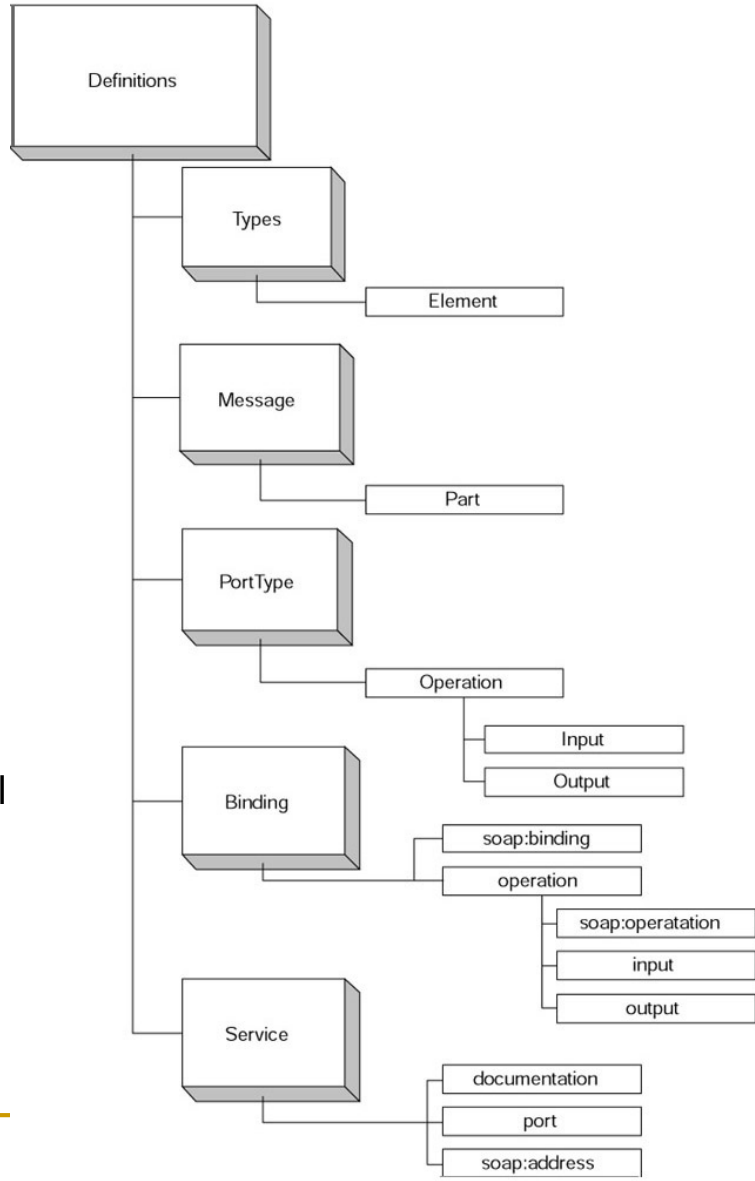
WSDL Grammar

- describe a collection of communication endpoints, called ***ports*.**
- Abstract parts:
  - The data being exchanged are specified as part of ***messages.***
  - Every type of action allowed at an endpoint is considered an ***operation.***
  - Collections of operations on an endpoint are grouped into ***port types***.
- The protocol and data format specifications for a particular port type are specified as a ***binding.***
  - Port: defined by associating a network address with a reusable binding
  - A common binding mechanism bring together all protocol and data formats with an message, operation, or endpoint.



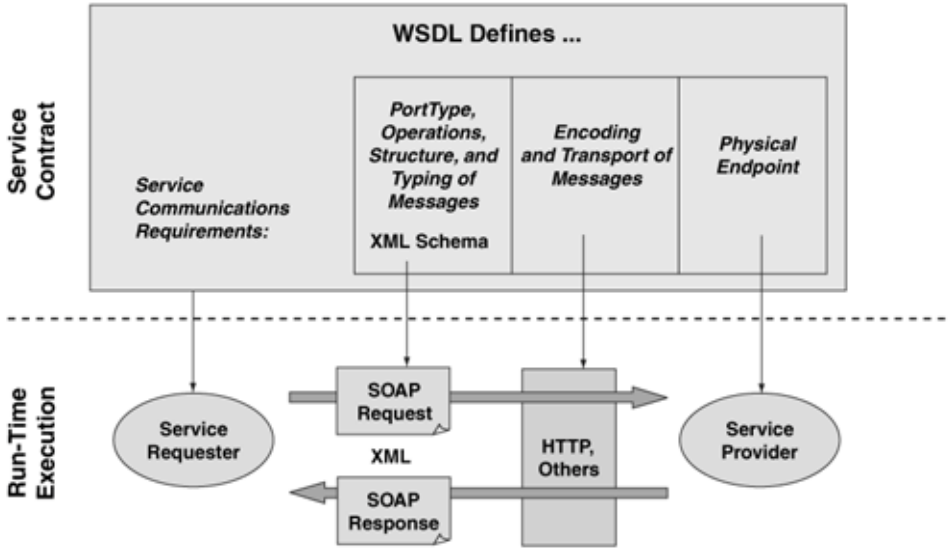
Parent Elements of an WSDL Document

- Definitions:
  - The root element of the WSDL document.
  - Defines the namespaces used for a particular description.
- Types:
  - Defines the elements and primitive types found in the XML of the SOAP request and response.
- Message:
  - Names the request and response messages.
- PortType:
  - Ties a particular request and response message to a particular service.
- Binding:
  - Indicates the type of transport used by the service.
  - Describes the contents of the message such as literal meaning to take the XML at face value, or image/gif.
- Service:
  - Names the services
  - Provides an element to document what the service actually accomplishes.



Usage of WSDL

- Definitions can be used to dynamically generate the SOAP messages that are exchanged to execute the service, illustrated using the request/response pattern.



Definitions - example

- is the root element of the WSDL document.
- Example: <definitions name="GetStockQuote></definitions>
  defines the name of the Web Service.
- Definitions tag will are needed to define several different namespaces to support the different primitive types and the types created by the GetStockQuote WS.
- Example:

```
<definitions name="GetStockQuote
targetNamespace="http://advocatemedia.com/GetStockQuote.wsdl"
  xmlns:myns = "http://advocatemedia.com/GetStockQuote.wsdl"
  xmlns:myXSD = "http://advocatemedia.com/GetStockQuote.xsd"
  xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
 </definitions>
```

  - targetNamespace: Defines the namespace for this document.
  - myns: A more precise definition for this document.
  - myXSD*:* Namespace for the schema types defined here.
  - xmlns:soap Namespace for the *SOAP* elements used in the document.
  - xmlns Sets the default namespace for *SOAP* elements.

Types - example

- defines the different elements used within the WS.
- the types element defines a schema so that a WSDL document is able to use the types defined in the schema standard rather than having to create its own.
- Example:

```
<types>
 <schema  targetNamespace="http://advocatemedia.com/GetStockQuote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="StockQuoteRequest">
       <complexType>
          <all>      <element name="symbol" type="string"/>      </all>
       </complexType>
    </element>
   <element name="StockQuoteResponse">
        <complexType>
          <all>        <element name="price" type="float"/>         </all>
       </complexType>
    </element>
  </schema>
 </types>
```

StockQuoteRequest / Response are the parent elements of the request and response documents.
The elements that actually contain values (i.e., symbol and price), are the children.
The types for these child elements are defined: symbol is defined as a string, price as a float.

Message - example

- Describe and name both the request and response message.
- Gives a path back to the types in the message.
- Example:

```
<message name="GetStockQuoteRequest">
    <part name="body" element="myXSD:StockQuoteRequest"/>
</message>
<message name="GetStockQuoteResponse">
    <part name="body" element="myXSD:StockQuoteResponse"/>
</message>
```

- The element definitions have the names of the parent elements in the *SOAP* document.
- The namespace myXSD is used as a prefix so that the application using the WSDL document knows to find the defs for these elements in the types portion of the doc
- This gives the request and response messages a name so the appls using this service know the name of the message to send & expect back when using a service
- This is protocol independent because there is no mention of HTTP or SMTP.
- The value of the name can be anything => select something meaningful!

PortType - example

- To use the two messages defined in the previous section with the message element, define them as the request and response for a particular service.
- This is done with the portType command:

```
<portType name="GetStockQuotePort">
    <operation name="GetStockQuote">
        <input message="myns:GetStockQuoteRequest"/>
        <output message="myns:GetStockQuoteResponse"/>
    </operation>
</portType>
```

GetStockQuote is the name of the operation.

  - The operation is the request for the price of a particular stock
  - The response is in the form of a price for that stock.
  - The input and output messages just combine the two definitions used earlier so that the client knows that a particular request and response message belongs to a particular method in a Web Service.
- A portType is an abstract definition of an interface.
  - It is abstract in the sense that it describes the operational interface of a service without going into the details of the data layout of the various parameters.
- A portType essentially consists of one or more operations, each consisting of several messages.

Binding - example

- Indicates the type of transport that a WS uses.
  - A binding describes how abstract defs of a portType are converted into a concrete representation
  - For example:
    - If an XML doc transmits its contents in the body, then the WSDL doc needs to define that.
    - If the document transmits its contents in Base64 encoding, then that would need to be defined
- There are two styles of the binding and the transport to choose from:
  - RPC: indicates that a message contains parameters and that there are return values.
  - Document: when the style attribute contains document, the request and response are passing XML documents within the body of the *SOAP* message.
- The namespace in the transport indicates which protocol is used : HTTP/SMTP
- Other infos:
  - input and output elements: defines the contents of the request and response messages.
  - Uses the contents of the document with/without any encoding (can be Base64, image/gif).
- Example:

```
<binding name="GetStockQuoteBindingName"  type="GetStockQuotePort ">
   <soap:binding    style="rpc"    transport= "http://schemas.xmlsoap.org/soap/http"/>
       <operation name="GetStockQuote">
        <soap:operation     soapAction="http://advocatemedia.com/GetStockQuote"/>
        <input>     <soap:body use="literal"/>   </input>
         <output>  <soap:body use="literal"/>   </output>
       </operation>
 </binding>
```

Defining the service - example

- defines the name of the service along with documentation and the location of the service.

- Example 1:

```
<service name=
    "AdvocateMediaGetStockQuotes">
<documentation>Simple Web Service to   Retrieve a stock quote</documentation>
<port name="GetStockQuotePort"   binding= "myns:GetStockQuoteBindingName">
  <soap:address  location="http://advocatemedia.com/GetStockQuote"/>
 </port>
</service>
```

The documentation element gives a developer the opportunity to provide some added information about what the Web Service accomplishes.

The soap:address names the Web Service as a whole.

Using import

- an alternate and perhaps easier way of writing the WSDL file.
- involves defining all the types in an *XML Schema Reduced* (XSD) file while putting all the other definitions relevant to the WS in the WSDL file.
- This way, the schema element and all of its children are in a separate file.
- If you are using the same elements in different Web Services, you can easily move the schema definitions from appl to appl.
- For example: there may be several stock-related WSs that use the same types and variables - one XSD file could support all the different services.

Ex. of GetStockQuote.xsd to be imported

```
<schema targetNamespace=
    "http://advocatemedia.com/GetStockQuote.xsd"
     xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="StockQuoteRequest">
     <complexType>
       <all> <element name="symbol" type="string"/> </all>
     </complexType>
    </element>
    <element name="StockQuoteResponse">
      <complexType>
       <all> <element name="price" type="float"/> </all>
      </complexType>
    </element>
</schema>
```

Simplified WSDL

```
<definitions name="GetStockQuote
targetNamespace="http://advocatemedia.com/GetStockQuote.wsdl
      "
    xmlns:myns =
     "http://advocatemedia.com/GetStockQuote.wsdl"
    xmlns:myXsd =
     "http://advocatemedia.com/GetStockQuote.xsd"
    xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap"
    xmlns="http://schemas.xmlsoap.org">

  <import
     namespace="http://advocatemedia.com/GetStocks/schemas"
     location ="http://advocatemedia.com/GetStocks/quote.xsd">

  <message name="GetStockQuote">
     <part name="body" element="myXSD:StockQuoteRequest"/>
  </message>
  <message name="GetStockQuoteResponse">
     <part name="body“
     element="myXSD:StockQuoteResponse"/>
  </message>

  <portType name="GetStockQuotePort">
        <operation name="GetStockQuote">
           <input message="myns:GetStockQuoteRequest"/>
           <output message="myns:GetStockQuoteResponse"/>
        </operation>
 </portType>
```

```
<binding name="GetStockQuoteBindingName"
          type="StockQuoteBinding">
   <soap:binding
          style="rpc"
          transport=" http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetStockQuote">
       <soap:operation
        soapAction="http://advocatemedia.com/GetStockQuote"/>
         <input>
            <soap:body use="literal"/>
         </input>
         <output>
           <soap:body use="literal"/>
         </output>
      </operation>
</binding>

<service name="AdvocateMediaGetStockQuotes">
      <documentation>Simple Web Service to
      Retrieve a stock quote</documentation>
  <port name="GetStockQuotePort"
        binding="myns:GetStockQuoteBindingName">
    <soap:address
      location="http://advocatemedia.com/GetStockQuote"/>
  </port>
</service>

</definitions>
```

SOAP

Simple Object Access Protocol (SOAP)

- enables objects not known to one another to communicate (exchange messages)
- similar to Internet Inter-ORB Protocol (IIOP) and Java Remote Method Protocol (JRMP)
  - instead using a binary data representation, it adopts text-based data representation using XML.
- independent of both the programming language & operational platform
- text format => firewall-friendly protocol.
- is backed by leading industrial players and can be expected to have universal support.
  - developed as a W3C recommendation.
  - version 1.2 from 2003.
  - SOAP become the *de facto* communication protocol standard for creating & invoking appls. exposed over a network.

Ensure interoperability and inter-appl communication

- defines a lightweight wire protocol & encoding format to represent data types, programming langs, & databases.
- can use a variety of Internet standard protocols (such as HTTP and SMTP) as its message transport,
- it provides conventions for representing communication models like
  - remote procedural calls (RPCs) and
  - document-driven messaging.
- named sometime also *Service-Oriented Access Protocol*
- Typical example:
  - allows communication between a Web appl. written in ASP.NET on a Windows Server communicating with a WS written in Perl on an Ubuntu server.

SOAP Defines …

1. an XML envelope for WS messages
   - SOAP envelope consists of a *body* and an optional *header*.
     - The SOAP body contained the application payload,
     - The SOAP header contained any other non-application data such as security, reliability, or transaction information.
2. a processing model for Web service intermediaries,
   - specified how network intermediaries could process the SOAP header information before delivering the SOAP body to the WS or client.
3. an encoding algorithm for serializing objects as XML.

SOAP encoding

- Motivated by a desire to create a distrib.obj. technology for the Web.
  - Earlier distributed object technologies such as CORBA, Java RMI, and DCOM failed to gain significant traction on the Internet.
  - As XML emerged, Microsoft proposed that it could be used as an architecturally neutral way to serialize graphs of objects=> SOAP was proposed as the carrier for these serialized object graphs and the serialization algorithm was dubbed *SOAP encoding*.
  - To use SOAP encoding, it is needed a between matching client and service implementations that understood the SOAP encoding algorithm.
  - The client and the server were assumed to both be implemented in conventional OOP languages!
    - One of the design principles behind XML is that it should be easily processable by a variety of applications.
    - SOAP encoding violates this principle.

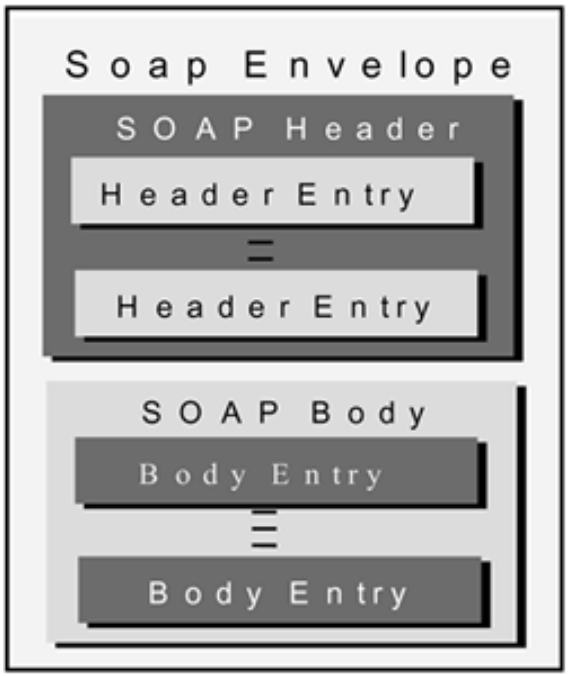
SOAP standard

- Dictates
  - how the XML looks within the *SOAP* document,
  - how the contents of that message are transmitted, and
  - how that message is handled at both the sender and receiver.
- Provides a standard set of vocabulary.
- Includes how SOAP messages should behave, different transports used, how errors get handled, and more.
- The terminology related to *SOAP* comes in two different categories:
  1. **transmission**
     - how *SOAP* messages relate to the protocol,
     - how the different *SOAP* nodes converse, and so on.
  2. **message**.
     - terms related to the XML within *SOAP*
- Also defines a small set of XML elements to encapsulate the data transmitted between nodes.

Elements

- a proper SOAP document: **envelope + body.**
- optional elements:

  header, fault, attachment
- The whole envelope—body plus header—is one complete XML document.
- The header entries
  - may contain information of use to recipients,
  - may also be of use to intermediate processors since they enable advanced features.
- The body, which contains the message contents, is consumed by the recipient.
- SOAP is agnostic about the message contents
  - the only restriction is that the message be in XML format.



Structure and examples

```
<?xml version="1.0" encoding="utf-8"?>

 <soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle=
    "http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
   <!-- Header information -->
</soap:Header>

<soap:Body>
    <!-- Body Information -->
</soap:Body>

</soap:Envelope>
```

- Request

```
<Envelope>
      <Body>
        <deposit>
          <amount
    type="int">700</amount>
        </deposit>
      </Body>
</Envelope>
```

- Response

```
<Envelope>
    <Body>
      <depositResponse/>
    </Body>
</Envelope>
```

SOAP envelope - example

- the primary container of a SOAP message's structure and is the mandatory element of a SOAP message.
- is represented as the root element of the message as Envelope.
- it is usually declared as an element using the XML namespace http://schemas .xmlsoap.org/soap/envelope/.

Code shows the SOAP envelope element in a SOAP message.

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     <!--SOAP Header elements - -/>
     <!--SOAP Body element - -/>
</SOAP-ENV:Envelope>
```

SOAP header - example

- represented as the first immediate child element of a SOAP envelope,
- has to be namespace qualified.
- may contain zero or more optional child elements, which are referred to as SOAP header entries.
- encodingStyle attribute will be used to define the encoding of the data types used in header element entries.
- actor attribute and mustUnderstand attribute can be used to indicate the target SOAP application node (Sender/Receiver/Intermediary) and to process the Header entries.

Code shows the sample representation of a SOAP header element in a SOAP message.

```
<SOAP-ENV:Header>
  <wiley:Transaction
            xmlns:wiley="http://jws.wiley.com/2002/booktx"
            SOAP-ENV:mustUnderstand="1">
      <keyValue> 5 </keyValue>
  </wiley:Transaction>
 </SOAP-ENV:Header>
```

SOAP header represents a transaction semantics entry using the mustUnderstand attribute.

The mustUnderstand attribute is set to "1", which ensures that the receiver (URI) of this message must process it.

SOAP body - example

- represents the mandatory processing information or the payload intended for the receiver of the message.
- A Body block of a SOAP message can contain any of the following:
  - RPC method and its parameters
  - Target application (receiver) specific data
  - SOAP fault for reporting errors and status information

Example: SOAP body representing an RPC call for getting the book price information from www.wiley.com for a book .

```
<SOAP-ENV:Body>
    <m:GetBookPrice xmlns:m="http://www.wiley.com/jws.book.priceList/">
    <bookname xsi:type='xsd:string'> Developing Java Web
    services</bookname>
    </m:GetBookPrice>
</SOAP-ENV:Body>
```

- SOAP Body can contain information defining an RPC call, business documents in XML, and any XML data required to be part of the message during communication.

SOAP attachments

- attachments in any data format that can be ASCII or binary (such as XML or non-text).

Example: Use "WileyCoverPage.gif" as an attachment and illustrates the use of the Content-ID (CID) reference in the body of the SOAP 1.1 message using absolute URI-referencing entities labeled for using Content-Location headers:

MIME-Version: 1.0 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml; start="<http://jws.wiley.com/coverpagedetails.xml>" Content-Description: SOAP message description.

```
--MIME_boundary-
    Content-Type: text/xml; charset=UTF-8
    Content-Transfer-Encoding: 8bit
    Content-ID: <http://jws.wiley.com/coverpagedetails.xml>
    Content-Location: http://jws.wiley.com/coverpagedetails.xml
 <?xml version='1.0' ?>
    <SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
       <SOAP-ENV:Body>
         <!-- SOAP BODY - ->
            <theCoverPage href="http://jws.wiley.com/DevelopingWebServices.gif"/>
         <!-- SOAP BODY - ->
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
 --MIME_boundary-
    Content-Type: image/gif
    Content-Transfer-Encoding: binary
    Content-ID: <http://jws.wiley.com/DevelopingWebServices.gif>
    Content-Location: http://jws.wiley.com/DevelopingWebServices.gif
<!--...binary GIF image... - ->
--MIME_boundary-
```

SOAP Fault

- is used to handle errors and to find out status information.

1. Faultcode:
   - defines the algorithmic mechanism for the SOAP application to identify the fault.
   - contains standard values for identifying the error or status of the SOAP application.

   The following fault-code element values are defined in the SOAP 1.1 specification:
   - VersionMismatch
   - MustUnderstand
   - Client
   - Server
2. Faultstring : provides a readable description of SOAP fault given by SOAP appl.
3. Faultactor: provides the info about the ultimate SOAP actor (Sender/Receiver/Intermediary) in the message who is responsible for the SOAP fault at the particular destination of a message.
4. Detail: provides the appl-specific error or status information related to Body block.

Exemplu:

```
<SOAP-ENV:Fault>
    <faultcode>Client</faultcode>
    <faultstring>Invalid Request</faultstring>
    <faultactor>http://jws.wiley.com/GetCatalog
      </faultactor>
</SOAP-ENV:Fault>
```

SOAP Encoding

- SOAP 1.1 specifications stated that SOAP-based applications can represent their data either as literals or as encoded values.
  - Literals refer to message contents encoded according to the XML Schema.
  - Encoded values refer to messages encoded based on SOAP encoding styles.
- The namespace identifiers for these SOAP encoding styles are defined in
  - http://schemas.xmlsoap.org/soap/encoding/ (SOAP 1.1) and
  - http://www.w3.org/2001/06/soap-encoding (SOAP 1.2).
- The SOAP encoding defines a set of rules for expressing its data types.
- It is a generalized set of data types that are represented by the programming languages, databases, and semi-structured data required for an application.
- SOAP encoding also defines serialization rules for its data model using an encodingStyle attribute under the SOAP-ENV namespace that specifies the serialization rules for a specific element or a group of elements.
- SOAP encoding supports both simple- and compound-type values.

SOAP Data Types and Structures: Primitive Types

Schema Primitive Type	Description	Example
xsd:int	signed integer value	-9 or 9
xsd:boolean	boolean whose value is either 1 or 0	1 or 0
xsd:string	string of characters	Rocky Mountains
xsd:float or xsd:double	signed floating point number (+,-)	-9.1 or 9.1
xsd:timeInstant	date/time	1969-05-07-08:15
SOAP-ENC:base64	base64-encoded information used for passing binary data within *SOAP* documents	SW89IjhhibdOI111QWgdGE

<int>98765</int> <decimal> 98675.43</decimal> <string> Java Rules </string>

Structs - examples

```
<xs:element name=”ShippingAddress”  xmlns:xs=”http://www.w3.org/2001/XMLSchema”>
<xs:complexType>
<xs:sequence>
    <xs:element ref=”Street”type=”xsd:string”/>
    <xs:element ref=”City” type=”xsd:string”/>
    <xs:element ref=”State” type=”xsd:string”/>
    <xs:element ref=”Zip” type=”xsd:string”/>
    <xs:element ref=”Country” type=”xsd:string”/>
  </xs:sequence>
  </xs:complexType>
</xs:element>
    ***
```

XML instance:

```
<e:ShippingAddress>
     <Street>1 Network Drive</Street>
     <City>Burlington</City>
     <State>MA</State>
     <Zip>01803</Zip>
     <Country>USA</Country>
</e:ShippingAddress>
```

Enumeration

- defines a set of names specific to a base type.
- Example of an enumeration data type expressed in a W3C XML Schema:

```
<xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”
        elementFormDefault=”qualified”>
      <xs:element name=”ProductType”>
        <xs:simpleType base=”xsd:string”>
          <xs:enumeration value=”Hardware”>
          <xs:enumeration value=”Software”>
        </xs:simpleType>
       </xs:element>
     </xs:schema>
```

Arrays - examples

```
<SymbolList
   SOAP-ENC: arrayType=
     "xsd:string[3]">
     <symbol>C</symbol>
     <symbol>GE</symbol>
     <symbol>DJI</symbol>
</SymbolList>
```

```
<AuthorInfo
  SOAP-ENC:arrayType="xsd:ur-type[4]" >
      <FirstName xsi:type="string">
       Brian</FirstName>
      <LastName xsi:type="string">
       Hochgurtel</LastName>
      <PhoneNumber xsi:type="int">
       3035551212</PhoneNumber>
      <BookTitle xsi:type="string">
       Cross Platform Web Services </BookTitle>
<AuthorInfo>
```

xsd:ur-type[4], indicates that there are four elements in the array of various types.

Example of an array of binary data

- is represented as text using base64 algorithms and expressed using a XML Schema

```
<myfigure xmlns:xsi=
        ”http://www.w3.org/2001/XMLSchema-instance”
   xmlns:enc=
        ” http://schemas.xmlsoap.org/soap/encoding”>
   xsi:type=”enc:base64”>
   sD334G5vDy9898r32323
</myfigure>
```

Partially transmitted arrays

- are defined using a SOAP-ENC:offset, which enables the offset position to be indicated from the first element (counted as zero-origin), which is used as an offset of all the elements that will be transmitted.

The following listing is an array of size [6];

using SOAP-ENC:offset=”4” transmits the fifth and sixth elements of a given array of numbers (0,1,2,3,4,5).

```
<SOAP-ENC:Array SOAP-ENC:arrayType=”xsd:string[6]”
    SOAP-ENC:offset=”[2]”>
    <item> No: 2</item>
    <item> No: 3</item>
    <item> No: 4</item>
    <item> No: 5</item>
</SOAP-ENC:Array>
```

Sparse arrays

- are defined using a SOAP-ENC:position, which enables the position of an attribute to be indicated with an array and returns its value instead of listing every entry in the array.

An example of using a sparse array in an array:

```
<SOAP-ENC:Array
  SOAP-ENC:arrayType=”xsd:int[10]”>
    <SOAP-ENC:int SOAP-ENC:position=”[0]”>
      0</SOAP-ENC:int>
    <SOAP-ENC:int SOAP-ENC:position=”[10]”>
      9</SOAP-ENC:int>
</SOAP-ENC:Array>
```

SOAP Communication

- SOAP supports the following 2 types of communication models:

1. SOAP RPC.
   - It defines a remote procedural call-based synchronous communication where the SOAP nodes send and receive messages using request and response methods and exchange parameters and then return the values.
2. SOAP Messaging.
   - It defines a document-driven communication where SOAP nodes send and receive XML-based documents using synchronous and asynchronous messaging.

SOAP RPC

- defines a tightly coupled communication model based on requests and responses.
- the SOAP message is represented by method names with zero or more parameters and return values.
- Each SOAP request message represents a call method to a remote object in a SOAP server
- the method calls are serialized into XML-based data types defined by the SOAP encoding rules.

SOAP Messaging

- represents a loosely coupled communication model based on message notification and the exchange of XML documents.
- The SOAP message body is represented by XML documents or literals encoded according to a specific XML schema, and it is produced and consumed by sending or receiving SOAP node(s).
- The SOAP sender node sends a message with an XML document as its body message and the SOAP receiver node processes it.

Transport bindings

- In addition to the XML in a *SOAP* request, there is also a header outside of the XML that is specific for the protocol being used, such as HTTP.
  - The information in this header contains the response code, the version of the protocol being used, the content type of the message, and perhaps other vendor-specific information.
- The SOAP specifications do not specify and mandate any underlying protocol for its communication as it chooses to bind with a variety of transport protocols between the SOAP nodes.
- According to the SOAP specifications for binding the framework, the SOAP bindings define the requirements for sending and receiving messages using a transport protocol between the SOAP nodes.
  - These bindings also define the syntactic and semantic rules for processing the incoming/outgoing SOAP messages and a supporting set of message exchanging patterns.
  - This enables SOAP to be used in a variety of applications and on OS platforms using a variety of protocols.

Preferences for transport bindings

- Although SOAP can potentially be used over a variety of transport protocols, initially the SOAP 1.0 specification mandated the use of HTTP as its transport protocol
- The later specifications opened their support for other Internet-based protocols like SMTP and FTP.
- Lately, major SOAP vendors have made their implementations available using popular transport protocol bindings like POP3, BEEP, JMS, Custom Message-Oriented-Middleware, and proprietary protocols using TCP/IP sockets.
- SOAP uses these protocol bindings as a mechanism for carrying the URI of the SOAP nodes.
  - Typically in an HTTP request, the URI indicates the endpoint of the SOAP resource where the invocation is being made.

SOAP over HTTP

- The use of HTTP as a transport protocol for SOAP communication becomes a natural fit for SOAP/RPC.
- Using SOAP over HTTP does not require overriding any existing syntactic and semantic rules of HTTP, but it maps the syntax and semantics of HTTP.
- By adopting SOAP over HTTP, SOAP messages can be sent through the default HTTP port 80 without requiring and opening other firewall ports.
- The only constraint while using SOAP over HTTP is the requirement to use the special header tag for defining the MIME type as Content-Type: text/xml.

SOAP over SMTP

- SOAP over *Standard Mail Transport Protocol* (SMTP) permits SOAP messages to be enabled with asynchronous communication and supports one-way notifications and document-driven messaging requirements.
  - a *SOAP* document needs to replace the HTTP header with info needed for e-mail.
  - Rather than having information needed for HTTP, such as Post and the URL of the service, the SMTP *SOAP* header contains an e-mail address, a subject, and a date.
  - A *SOAP* request may also contain unique message Ids. This is just like sending an e-mail to a person except that software generated the e-mail to send to the receiver.
  - Instead of a text message, the application sends the *SOAP* document that contains the XML needed for e.g. a service.
  - The SOAP message will send the request with a Message-Id in the header and the response SOAP message will contain an In-Reply-To header containing the originator's Message-Id.

Other SOAP Bindings

- **SOAP over HTTP/SSL.**
  - take advantage of using Secure Socket Layer (SSL) for security and other HTTP-based protocol features.
  - enables encryption of messages with greater security & confidentiality between the SOAP nodes.
  - Is possible to add MAC (Media access control) addresses of network card interfaces in messages.
- **SOAP over JMS.**
  - To enable SOAP messages to communicate with J2EE-based components and messaging applications,
  - most SOAP vendors provide SOAP messaging over JMS (Java Messaging Service) with JMS-compliant MOM providers such as Sun One MQ, Sonic MQ, Websphere MQSeries, and so on.
  - allows SOAP-based asynchronous messaging and enables the SOAP messages to achieve reliability and guaranteed message delivery using a JMS provider.
  - the JMS destination queues are represented in the SOAP messages as target destinations.
  - SOAP nodes use the JMS queue for sending and receiving SOAP requests and SOAP responses.
- **SOAP over BEEP.**
  - Blocks Extensible Exchange Protocol (BEEP) defines a generic application transport protocol framework for connection-oriented, asynchronous messaging that enables peer-to-peer, client-server, or server-to-server messaging.
  - enables SOAP developers to focus on the aspects of the SOAP applications instead of finding a way to establish communication: BEEP takes care of the communication protocol.
  - BEEP, as a protocol, governs the connections, authentication, and sending and receiving of messages at the level of TCP/IP.

SOAP security

- SOAP messages do not carry or define any specific security mechanisms.
- using the SOAP headers provides a way to define and add features enabling the implementation of application-specific security in a form of XML-based metadata.
  - the metadata information can be application-specific information incorporating message security with associated security algorithms like encryption and digital signatures.
- SOAP supports various transport protocols for communication, thus it also is possible to incorporate transport protocol-supported security mechanisms like SSL/TLS for SOAP messages.
- all of the security-related elements are identified using a single namespace identifier using the prefix SOAP-SEC and with an associated URI using http://schemas.xmlsoap.org/soap/security/.
  - It defines the three security element tags <SOAP-SEC: Encryption>, <SOAP-SEC:Signature>, and <SOAP-SEC:Authorization>.
    - Use of these security tags enables the incorporation of encryption, digital signatures, and authorization in SOAP messages.

UDDI

Universal Discovery, Description & Integration

- defines a standard way for registering, deregistering, and looking up WS.
- announced in 2000 as the joint work of Microsoft, IBM, and Ariba.
- The specifications as well as links to other resources are available at http://www.uddi.org
- UDDI directories are not limited to WS, and can contain services based on a number of protocols and technologies such as telephone, FTP, email, CORBA, SOAP,Java RMI.
- There are two main parts to UDDI:
  - the specification for how to hold all of the information, and
  - the implementation of the specification.

Registry Standards

- UDDI is a standards-based specification for Web service registration, description, and discovery.
- Similar to a telephone system's yellow pages, a UDDI registry's sole purpose is to enable providers to register their services and requestors to find services.
  - Once a requestor finds a service, the registry has no more role to play between the requestor and the provider.
- UDDI specification defines
  - an XML schema for SOAP messages and
  - APIs for applications wanting to use the registry.
    - A provider registering a WS with UDDI must furnish business, service, binding, and technical information about the service.

APIs

- A repository can be contacted in two ways:
  - either through a Web browser or
  - through *SOAP* requests and responses by using a particular API.
- UDDI specification includes two categories of APIs for accessing UDDI services from applications:

1. Inquiry APIs— enable lookup and browsing of registry information
2. Publishers APIs— allow applications to register services with the registry

- UDDI APIs behave in a synchronous manner.
- UDDI uses SOAP as the base protocol.
- Note that UDDI is a specification for a registry, not a repository.
  - As a registry it functions like a catalog, allowing requestors to find available services.
  - A registry is not a repository because it does not contain the services itself.

UDDI operator

- offers a registry for general public use.
- has to offer a conformant interface to its registry.
- can offer extra services to its client
  - the UDDI specification mandates that the core registry described by the information model is an exact replica of other operator's registries.
  - E.g. offer a Web-based interface to their registries that facilitates service discovery at design time.

UDDI web sites

- UDDI Web sites are supported by a group of industry leaders, including HP, IBM, Microsoft, and SAP, who formed a consortium to maintain the *UDDI registry system*.
  - This system is a database of businesses and the URL of the Web Services they offer, any available WSDL files, and provided business information.
  - Each of the vendors replicates the entries it receives back to the others.
- Implementations of UDDI Version 1:
  - Microsoft http://uddi.microsoft.com/
  - IBM http://www-3.ibm.com/services/uddi/find
- Implementations of UDDI Version 2
  - Microsoft https://uddi.rte.microsoft.com/search/frames.aspx
  - IBM https://www3.ibm.com/services/uddi/v2beta/protect/registry.html
  - Hewlett Packard https://uddi.hp.com/uddi/index.jsp
  - SAP https://websmp201.sap-ag.de/~form/uddi_discover