أكاديمية الحكومة الإلكترونية الفلسطينية
The Palestinian eGovernment Academy
www.egovacademy.ps

Tutorial III:

Process Integration and Service Oriented Architectures

Session 8
WSDL

Prepared By

Mohammed Aldasht

European Commission
TEMPUS

About

This tutorial is part of the PalGov project, funded by the TEMPUS IV program of the Commission of the European Communities, grant agreement 511159-TEMPUS-1-2010-1-PS-TEMPUS-JPHES. The project website: www.egovacademy.ps

Project Consortium:

Birzeit University, Palestine (**Coordinator**)	University of Trento, Italy
Palestine Polytechnic University, Palestine	Vrije Universiteit Brussel, Belgium
Palestine Technical University, Palestine	Université de Savoie, France
Ministry of Telecom and IT, Palestine	University of Namur, Belgium
Ministry of Interior, Palestine	TrueTrust, UK
Ministry of Local Government, Palestine	

Coordinator:
Dr. Mustafa Jarrar
Birzeit University, P.O.Box 14- Birzeit, Palestine
Telfax:+972 2 2982935 mjarrar@birzeit.edu

© Copyright Notes

Tutorial Map

Intended Learning Objectives

A: Knowledge and Understanding

- 3a1: Demonstrate knowledge of the fundamentals of middleware.
- 3a2: Describe the concept behind web service protocols.
- 3a3: Explain the concept of service oriented architecture.
- 3a4: Explain the concept of enterprise service bus.
- 3a5: Understanding WSDL service interfaces in UDDI.

B: Intellectual Skills

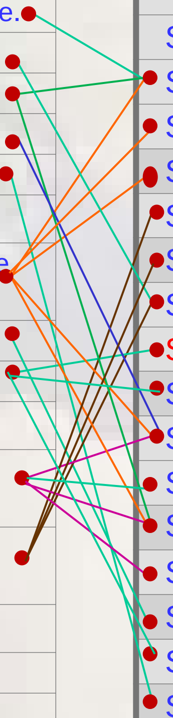
- 3b1: Design, develop, and deploy applications based on Service Oriented Architecture (SOA).
- 3b2: use Business Process Execution Language (BPEL).
- 3b3: using WSDL to describe web services.

C: Professional and Practical Skills

- 3c1: setup, Invoke, and deploy web services using integrated development environment.
- 3c2: construct and use REST and SOAP messages for web services communication.

D: General and Transferable Skills

- d1: Working with team.
- d2: Presenting and defending ideas.
- d3: Use of creativity and innovation in problem solving.
- d4: Develop communication skills and logical reasoning abilities.



Title	T	Name
Session0: Syllabus and overview	0	Aldasht
Sesson1: Introduction to SOA	2	Aldasht
Session2: XML namespaces & XML schema	2	Aldasht
Session 3: Xpath & Xquery	4	Romi
Session4: REST web services	3	M. Melhem
Session5: Lab2: Practice on REST	3	M. Melhem
Session 6: SOAP	2	Aldasht
Session 7: WSDL	3	Aldasht
Session8: Lab 3: WSDL practice	3	Aldasht
Session9: ESB	4	Aldasht
Session10: Lab4: Practice on ESB	4	Aldasht
Session11: integration patterns	4	M. Melhem
Session12: Lab5: integration patterns	4	M. Melhem
Session13: BPEL	3	Aldasht
Session14: Lab6: Practice on BPEL	3	Aldasht
Session15: UDDI	2	Aldasht

Session 8: WSDL

Session ILOs

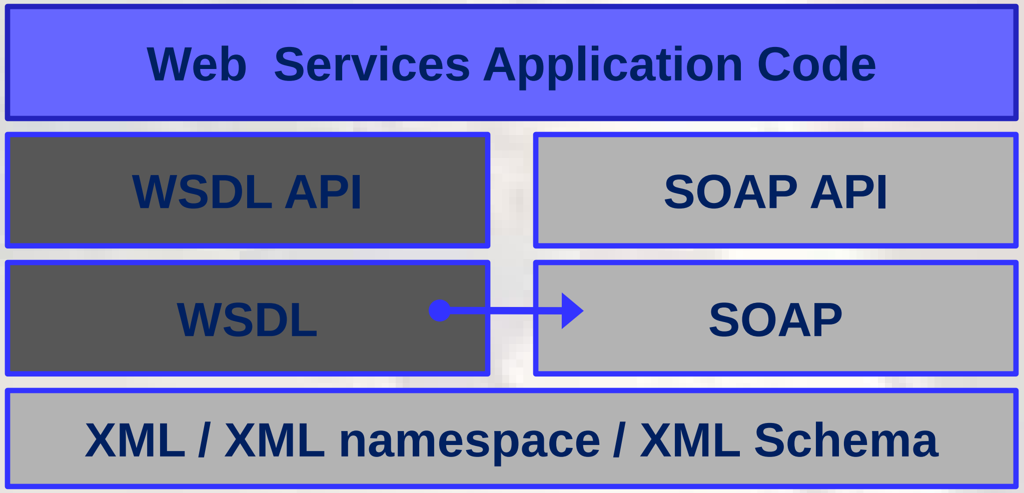
After completing this session students will be able to use WSDL to describe web services.

Session Outlines

- **Introduction**
- WSDL building blocks
- Structure of a WSDL document
- The SOAP binding

Overview

- A consumer first examines the WSDL description before requesting a service.



Web Services Application Code
WSDL API
SOAP API
WSDL
SOAP
XML / XML namespace / XML Schema

Source, [3]

The Palestinian e-government academy

Introduction I

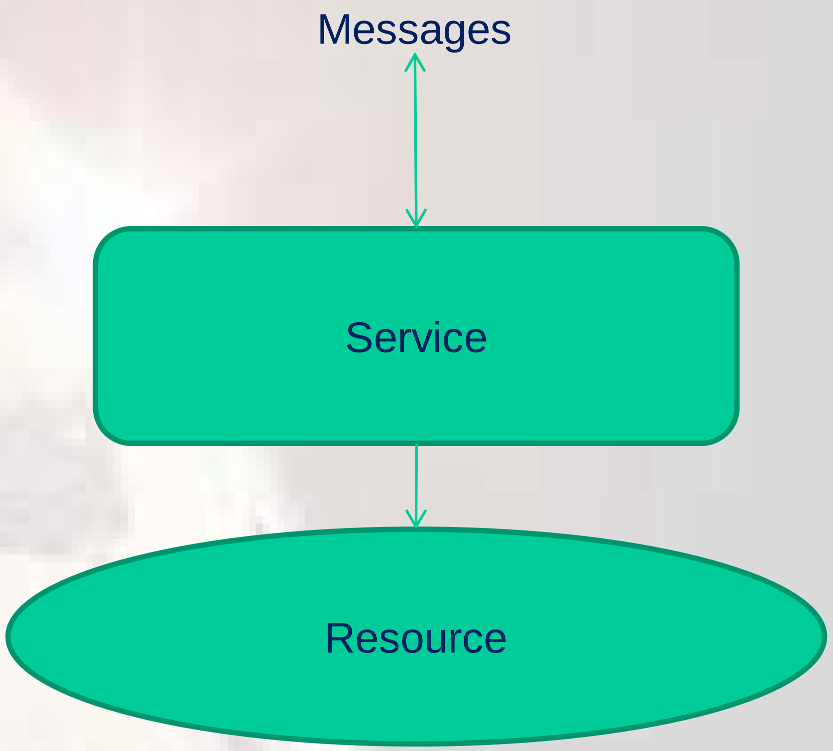
- **Web Services Description Language** (WSDL): is an XML-based language that provides a model for describing Web services [2].
- WSDL is often used in combination with SOAP and an XML Schema to provide Web services over the Internet.
  - A client program connecting to a Web service can read the WSDL file to determine what operations are available on the server.
  - Any special datatypes used are embedded in the WSDL file in the form of XML Schema.
  - The client can then use SOAP to actually call one of the operations listed in the WSDL file using XML or HTTP.

Introduction II

- Web service refers to the piece of code implementing the XML interface to a resource
- This enables consumer with XML support to integrate with Web service applications
- XML Schema
  - Allows developers to describe the structure of XML messages
  - But, can't describe the additional details involved in communication with a Web service

Introduction III

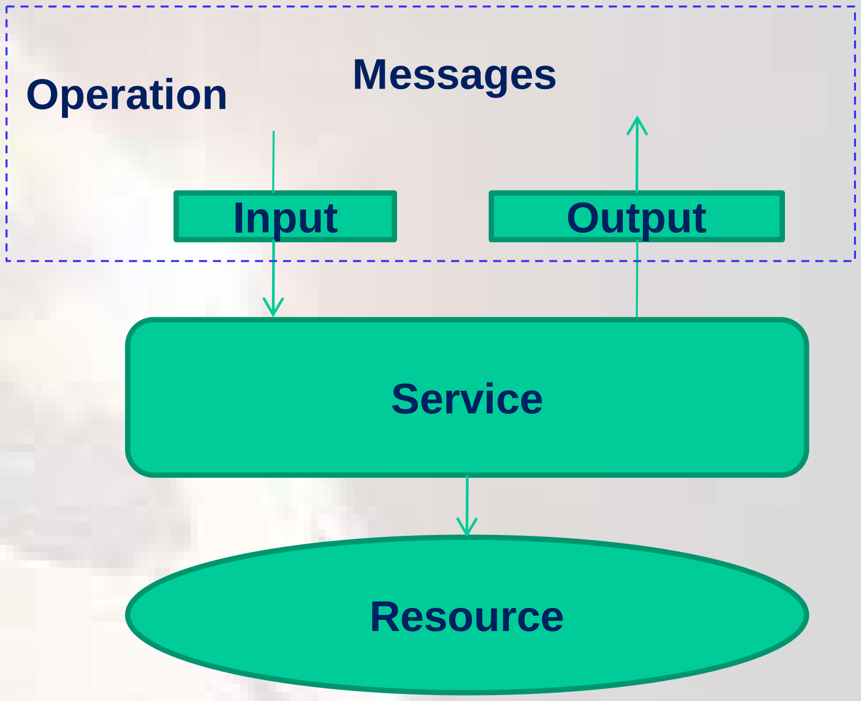
- A message exchange is referred to as an operation.
- Operations are what consumers care about most since they're the focal points of interaction.
- Consumers must be aware of the groupings of related operations into interfaces, since it impacts the way they write their code service [1]



Communicating with a web service , Source, [1]

Introduction IV

- Consumers must be aware of these groupings since it impacts the way they write their code.
- Consumers must also know what communication protocol to use for sending messages to the service [1].
  - Also, specific mechanics involved in using the given protocol such as the use of commands, headers, and error codes



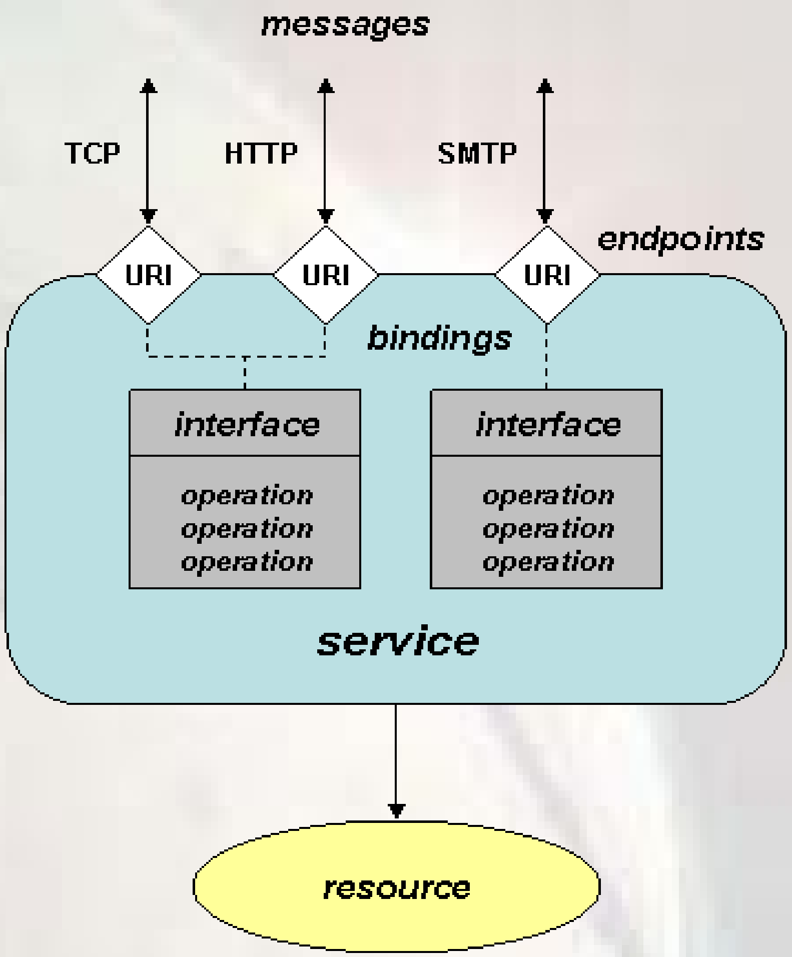
Operations (message exchange), Source, [1]

Introduction V

- A binding influences the way abstract messages are encoded on the wire by specifying the style of service.
  - Document vs. RPC
  - The encoding mechanism, literal vs. encoded.
- A service can support multiple bindings for a given interface.
- Each binding should be accessible at a unique address identified by a URI, also referred to as a Web service endpoint [1].

Introduction VI: Interfaces and Bindings

Interfaces and Bindings, Source, [1]



Session Outlines

- Introduction
- **WSDL building blocks**
- Structure of a WSDL document
- The SOAP binding

WSDL building Blocks

- WSDL separates a service into two parts [3]:
  - Abstract interfaces: unfolds the operations supported by the web service, the operation parameters and abstract data types.
    - Independently from, any concrete network address, communication protocol or data structure.
  - Concrete implementation: binds the abstract interface description to a concrete network address, to a protocol and to a concrete data structure.
    - A web service consumer can bind to such an implementation and invoke a service.

WSDL building Blocks

- A WSDL document is an XML instance composed of elements.
- Elements are declared in a WSDL schema definition with target namespace: `http://schemas.xmlsoap.org/wsdl/`.
- See next slide for:
  - An initial WSDL interface description for the `getPhoneNumber` service, Source, [3].

getPhoneNumber service

```
<definitions name = “phoneNumberService”
  targetNamespace = “http://companyx.com/ns/phoneNumber/wsdl”
 xmlns:tns=“http://companyx.com/ns/phoneNumber/wsdl”
 xmlns:SOAP-EXT=“http://schemas.xmlsoap.org/wsdl/soap/”
 xmlns=“http://schemas.smlsoap.org/wsdl/”>
   <types>
       <schema
          targetNamespace=“http://companyx.com/ns/phoneNumber/wsdl”
           xmlns=“http://www.w3.org/2001/XMLSchema”>
              <complexType name=“NameInType”>
           <sequence>
                <element name=“first-name” type=“xsd:string”/>
                <element name=“last-name” type=“xsd:string”/>
            </sequence>
               </complexType>
           </schema>
       </types>
        <message name=“GetPhoneNumberIn”>
               <part name=“name” type=“tns:NameInType”/>
         </message >
         <message name=“GetPhoneNumberOut”>
             <part name=“return” type=“xsd:string”/>
        </message >
        <portType name=”PhoneNumberPortType”>
             <operation name=“getPhoneNumber”>
                  <input message=“tns:GetPhoneNumberIn”/>
                  <output message=“tns:GetPhoneNumberOut”/>
             </operation>
         </portType>
```

Abstract data type definitions

Data that is returned

Port type containing one operation

An operation with input and output message

Summary of elements collectively describing a web service interface

- `portType`: a named collection of operations
- `operation`: abstractly describes a service call.
  - May contain `input` message, `output` message and optionally several `fault` message.
- `message`: an abstract description of the data that is sent.
  - Messages consist of logical units called parts.

Summary of elements collectively describing a web service interface

- `part`: is associated with a data type.
- `type`: provides a container for data type definitions.
  - WSDL 1.1 refer to XML schema data types, but doesn't prohibit the use of other type systems.
- See next slide for:
  - A WSDL implementation description for the `getPhoneNumber` service, Source, [3].

```
<binding name="PhoneNumberSoapBinding"  type="tns:PhoneNumberPortType">
  <SOAP-EXT:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getPhoneNumber">
    <SOAP-EXT:operation soapAction=""/>
    <input>
      <SOAP-EXT:body use="encoded"/>
    </input>
    <output>
      <SOAP-EXT:body use="encoded"/>
    </output>
  </operation>
</binding>
<service name="PhoneNumberService">
  <port name="PhoneNumberPort" binding="tns:PhoneNumberSoapBinding">
    <SOAP-EXT:address
     location="http://www.companyx.com/servlet/rpcrouter"/>
 </port>
</service>
</definitions>
```

Use SOAP RPC style

Bind an abstract operation to this concrete implementation …

… and map the abstract input and output messages to these concrete messages

Service name

Network address of service

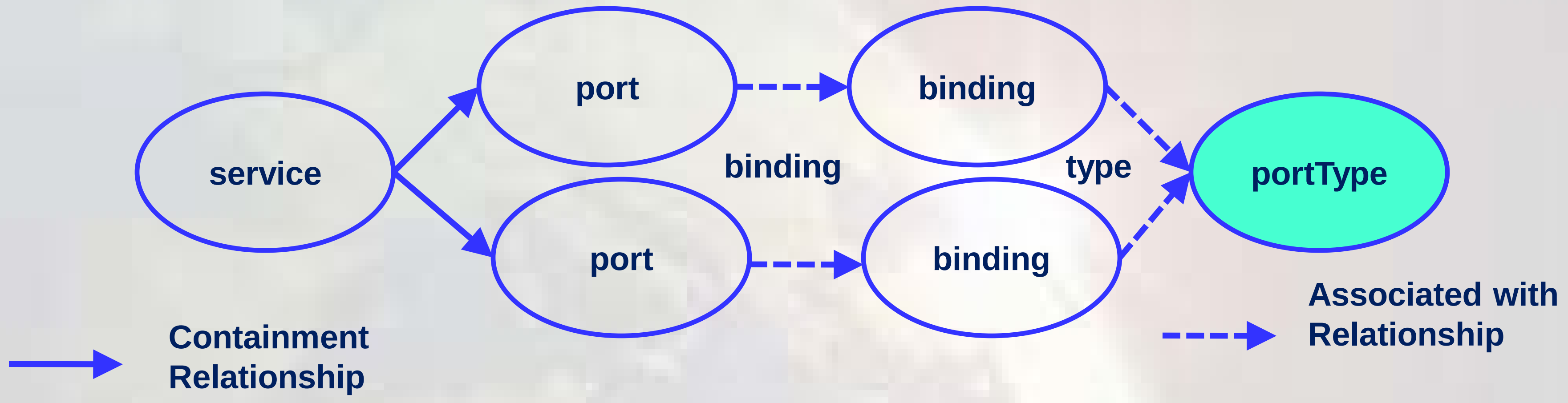
Summary of elements describing a web service implementation

- `binding`: provides concrete protocol and data format specifications for a particular port type.
- `port`: describes the network address of a service.
  - In combination with a binding it provides info about concrete service endpoint.

Summary of elements describing a web service implementation

- `service`: a collection of related ports.
  - May share the same port type, but employ different bindings or network addresses.
  - May be concrete service endpoint implementations of several port types.

WSDL service interface and service implementation description



Source, [3]

WSDL related name spaces

- Although the use of other type systems is not forbidden, WSDL uses XML schema definitions and XML schema instance attributes from representing abstract data types:
  - Namespace prefix xsd refers to URI `http://www.w3.org/2001/XMLSchema`.
  - Namespace prefix xsi refers to URI `http://www.w3.org/2001/XMLSchema-instance`

WSDL related name spaces, cont.

- Additional namespaces with WSDL 1.1 specification:
  - Namespace prefix wsdl refers to URI `http://schemas.xmlsoap.org/wsdl`
  - Namespace prefix SOAP-EXT refers to URI `http://schemas.xmlsoap.org/wsdl/soap`

Session Outlines

- Introduction
- WSDL building blocks
- **Structure of a WSDL document**
- The SOAP binding

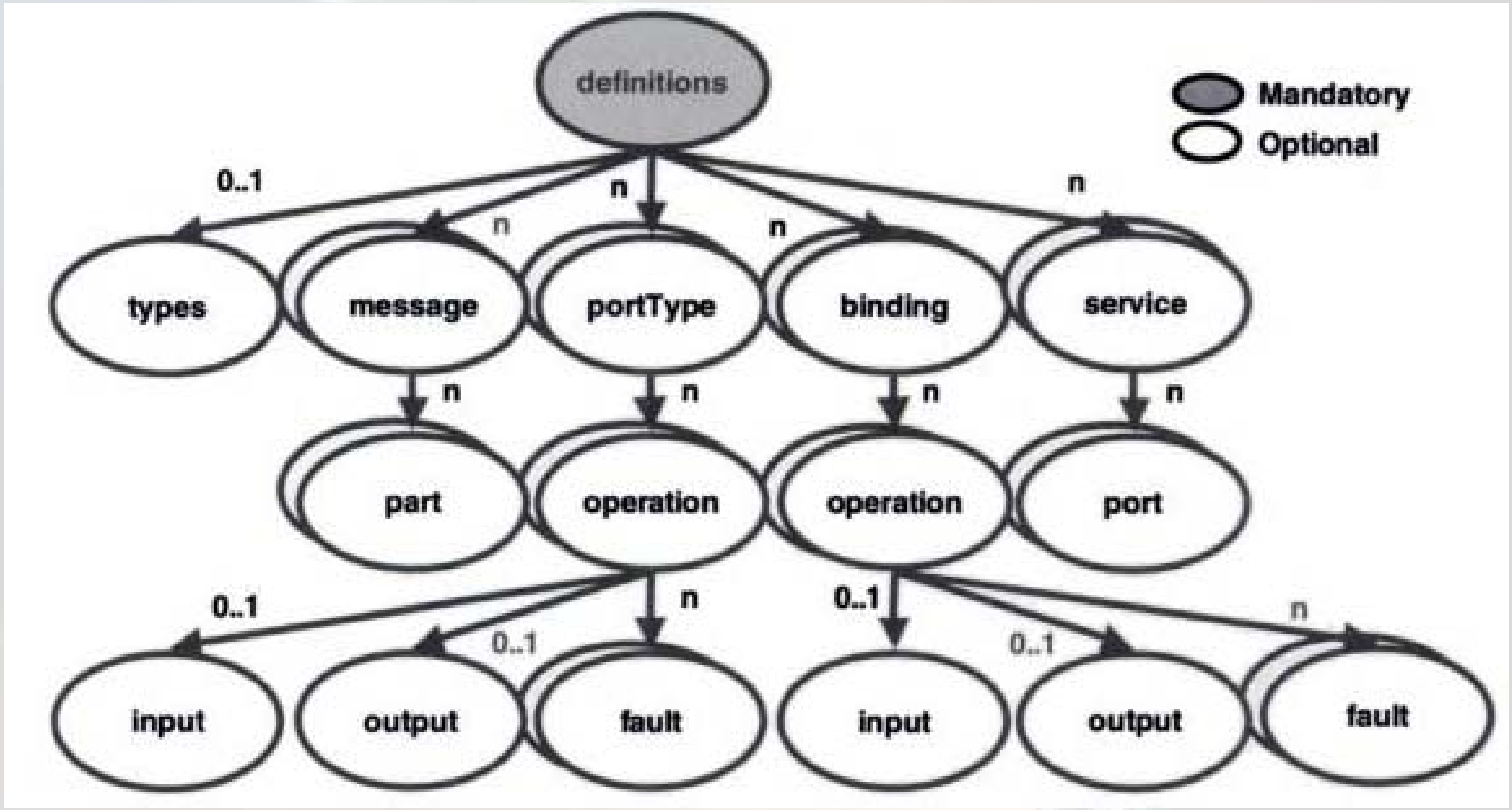
WSDL document containment structure

- Three important containment structure elements: (see next slide)
  - `definitions`: a container for all WSDL elements.
  - `portType`.
  - `binding`: each binding related to a distinct port type maps a concrete protocol implementation to this port type.

WSDL document containment structure

- Message provides a data types container for operations via its ports.
- Individual data types may appear multiple times in several messages.
- A specific input message may be identical, and reusable for several applications.
- Bindings might be reusable, also.

WSDL document containment structure



Source, [3]

Service interface-related elements of WSDL document

- `definitions` element: may have `name`, `targetNamespace` and `xmlns`.
- `types` element: a container for all data type definitions that belong to the messages.
- `message` element: represents operation parameters for one interaction between a client and a service. e.g:
  - Input parameters for a SOAP RPC are described as a message.
  - Another message presents the return and output parameters for this call.
- `part` element: contained in message element; may carry `name` attribute, `element` attribute and `type` attribute.

Definitions elements

- `wsdl` namespace prefix is defined by default.
- Identical namespaces assigned to the `targetNamespace` attribute and the `tns` namespace prefix.
- The target namespace of local WSDL definitions together with the `tns` namespace prefix helps a processor to locate these local definitions when they are referred to.

Definitions example

```
<definitions name=“phoneNumberService”
 targetNamespace=“http://companyx.com/ns/phoneNumber/wsdl”
 xmlns:tns=“http://companyx.com/ns/phoneNumber/wsdl”
 xmlns:SOAP-EXT=“http://schemas.xmlsoap.org/wsdl/soap/”
 xmlns:”http://schemas.smlsoap.org/wsdl”>
  <documentation>
    WSDL document describing a phone number service
  </documentation>
  <!-- The WSDL specification goes here. -->
  ...
</definitions>
```

Types element

- The types element is a container for all abstract data type definitions that belong to messages.
- Message parts link types to messages.
- WSDL prefers the use of XML schema for the type definitions
- In the following example all elements are based on the XML schema built-in `string` type.

The Palestinian e-government academy

Types Example

```
<types>
   <schema

   targetNamespace=“http://companyx.com/ns/phoneNu
   mber/wsdl”
    xmlns=“http://www.w3.org/2001/XMLSchema”>
       <element name=“FirstNameIN”
   type=“xsd:string”/>
       <element name=“LastNameIN”
   type=“xsd:string”/>
       <element name=“PhoneNumber”
   type=“xsd:string”/>
       <element name=“NoNameFound”
   type=“xsd:string”/>
   </schema>
</types>
```

Message and `part` elements

- As mentioned earlier, a message represents operation parameters for one interaction between a client and a service.
- See next slide for an example of three messages, each holding one part, respectively:

Message and part example

```
<message name=“GetPhoneNumberIn”>
  <part name=“name” element=“tns:NameIn”/>
</message>
<message name=“GetPhoneNumberOut”>
  <part name=“return” element=“tns:PhoneNumber”>
</message>
<message name=“NoNameFoundOut”>
  <part name=“failure” element=“tns:NoNameFound”/>
</message>
```

Message and part example, cont.

- The element attribute refers to the element declarations in the types container.
- Alternatively, we can have defined the data type of each part via the type attribute.

```
<message name=“GetPhoneNumberIn”>
  <part name=“name” type=“tns:NameInType”/>
</message>
<message name=“GetPhoneNumberOut”>
  <part name=“return” type=“xsd:string”>
</message>
<message name=“NoNameFoundOut”>
  <part name=“failure” type=“xsd:string”/>
</message>
```

Mapping of a WSDL part to a SOAP RPC style message

- See the message in the next slide.
- Dotted arrows illustrate how a part name of the WSDL document describing the phone number service is mapped to a procedure call parameter accessor in a SOAP RPC style message body to invoke this service.

WSDL part to a SOAP RPC style message

WSDL Document:

```
<message name="GetPhoneNumberIn">
  <part name="name" type="tns:NameIn"/>
</message>
```

SOAP Message Body (RPC Style):

```
<SOAP-ENV:Body>
  <getPhoneNumber ...>
    <name>
      <first-name xsi:type="xsd:string">Ahmad M.</first-name>
      <last-name xsi:type="xsd:string">Ahmad</last-name>
    </name>
   </getPhoneNumber>
</SOAP-ENV:Body>
```

Source, [3]

Mapping of a WSDL part to a SOAP document style message

- Here, parts point to the XML instance contained in the SOAP message body.
- No additional wrapper.
- The element representing the XML instance goes directly into the SOAP message body element.
- See next slide!

WSDL part to a SOAP document style message

WSDL Document:

```
<message name="GetPhoneNumberIn">
  <part name="name" element="tns:NameIn"/>
</message>
```

SOAP Message Body (Document Style):

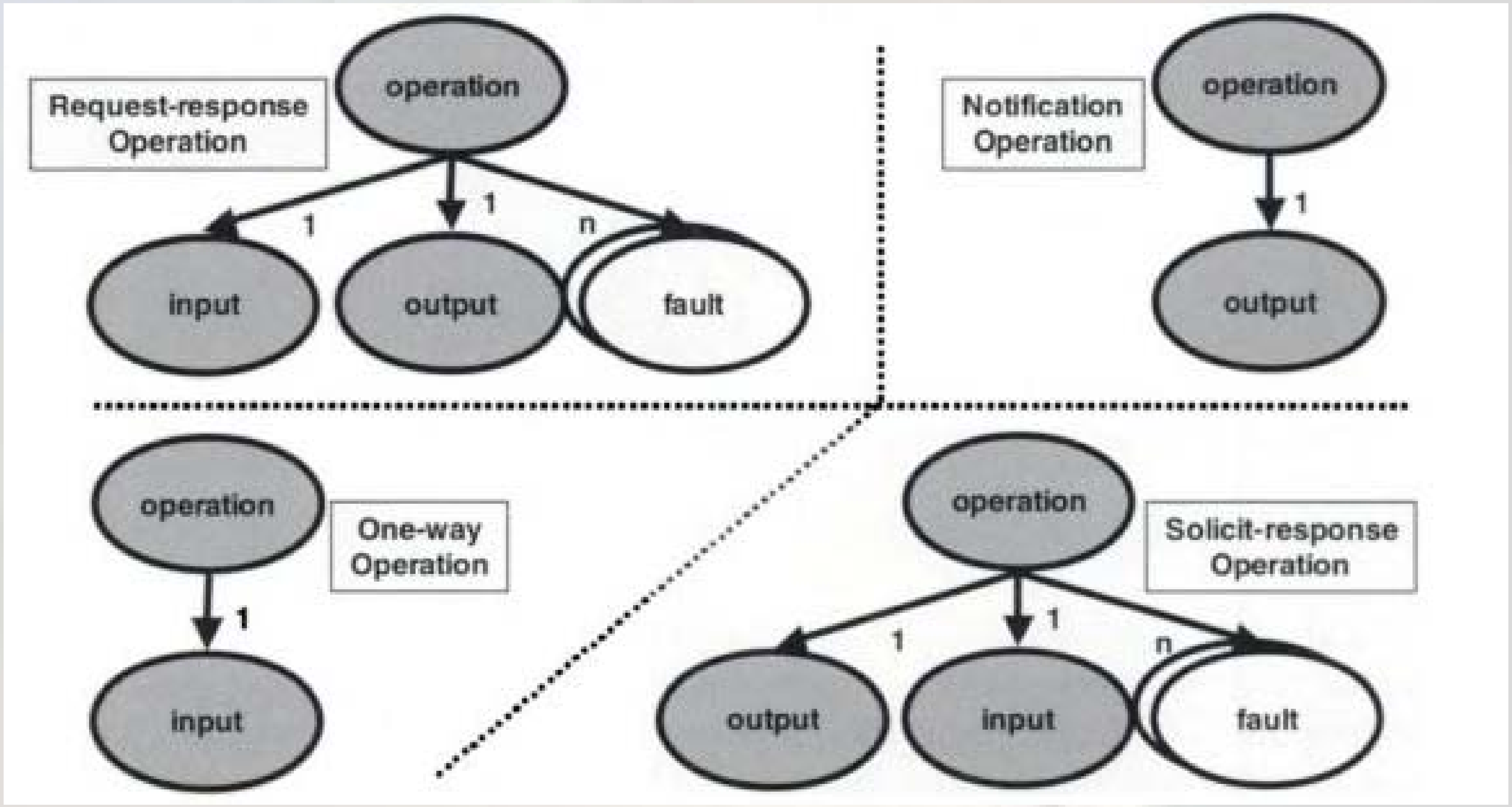
```
<SOAP-ENV:Body>
  <NameIn>
      <first-name xsi:type="xsd:string">Ahmad M.</first-
name>
      <last-name xsi:type="xsd:string">Ahmad</last-name>
  </NameIn>
</SOAP-ENV:Body>
```

Source, [3]

The `operation` element

- 4 operation types:
  - One-way: has only an `input` element specifying the operation message.
  - Request-response: contains one `input`, one `output` and optionally, one or more `fault` elements.
  - Solicit-response: contains one `input`, one `output` and optionally, one or more `fault` elements.
  - Notification: contains only an output element specifying the operation message.

operation element



Source, [3]

Mapping of a WSDL operation to a SOAP RPC style message

- An operation element may hold two attributes:
  - `name`: it is mandatory and specifies the operation name
  - `parameterOrder`: carries a space-separated list of message parts.
    - May be used for request-response and solicit-response operations with an RPC binding to save the function call signature.

WSDL operation to a SOAP RPC style message

WSDL Document:

```
<portType name=“PhoneNumberPortType”>
  <operation name=“getPhoneNumber”>
    <input message=“tns:GetPhoneNumberIn”/>
    <output message=“tns:PhoneNumberOut”/>
  </operation>
</portType>
```

SOAP Message Body (RPC Style):

```
<SOAP-ENV:Body>
  <getPhoneNumber ...>
      ...
   </getPhoneNumber>
</SOAP-ENV:Body>
```

Source, [3]

Binding-related elements of a WSDL document

- `binding` element almost matches the structure of `portType` element.
- `binding` must map the abstract `portType` description to a concrete implementation, carries two attributes:
  - `name`: is mandatory, and specifies the name of the binding
  - `type`: is mandatory, and refers to the port type that it binds.

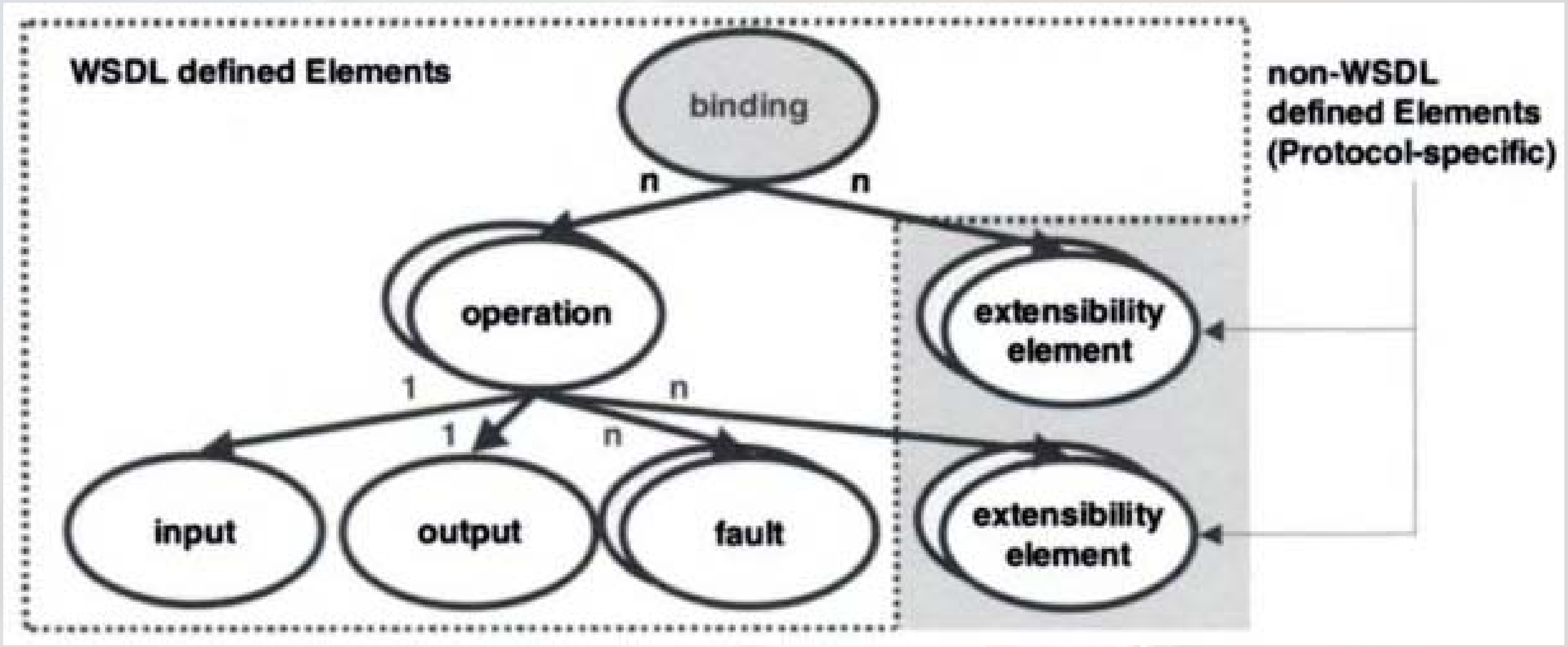
binding example, Source, [3]

```
<binding name="PhoneNumberSoapBinding"
         type=tns:PhoneNumberPortType">
  <!--  SOAP binding extensibility element        -->
  <SOAP-EXT:binding style="rpc">
   transport="http://schemas.xmlsoap.org.soap/http"/>
 <operation name="getPhoneNumber">
 <!--  SOAP operation extensibility element       -->
   <SOAP-EXT:operation soapAction=""/>
   <input>
     <SOAP-EXT:body use="encoded"/>
   </input>
   <output>
     <SOAP-EXT:body use="encoded"/>
   </output>
 </operation>
</binding>
```

The binding element

- `binding` type attribute together with the name attribute uniquely identifies a binding.
- Since various bindings may refer to the same port type.

binding element details



Source, [3]

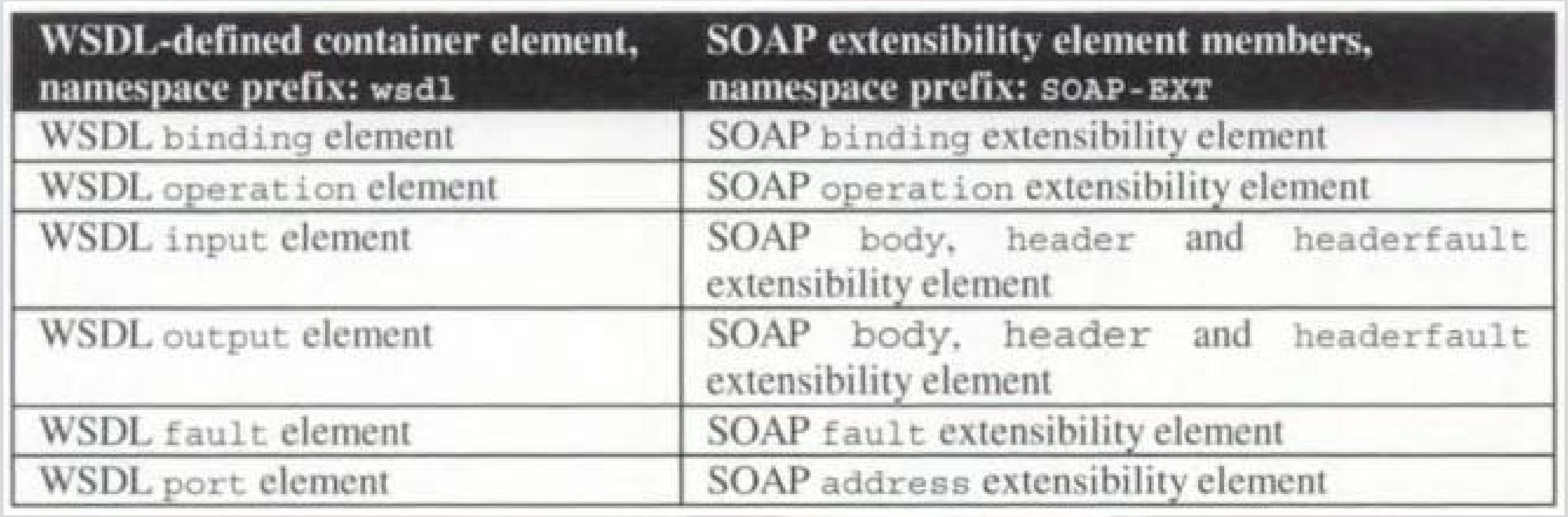
Session Outlines

- Introduction
- WSDL building blocks
- Structure of a WSDL document
- **The SOAP binding**

The SOAP binding

- Binding to SOAP 1.1, means that we must fill these containers with SOAP 1.1 extensibility elements.
- Providing a distinct WSDL binding requires the definition of extensibility elements for the `binding`, the `operation` and the `port` elements.
- In our examples, the namespace prefix for SOAP binding is SOAP-EXT.
- The table contains WSDL-defined binding elements and contained SOAP extensibility element.

WSDL-defined binding elements and contained SOAP extensibility element



WSDL-defined container element, namespace prefix: wsdl	SOAP extensibility element members, namespace prefix: SOAP-EXT
WSDL binding element	SOAP binding extensibility element
WSDL operation element	SOAP operation extensibility element
WSDL input element	SOAP body, header and headerfault extensibility element
WSDL output element	SOAP body, header and headerfault extensibility element
WSDL fault element	SOAP fault extensibility element
WSDL port element	SOAP address extensibility element

Source, [3]

SOAP extensibility elements

- SOAP `binding` extensibility element: has two attributes:
  - `transport`: indicates the transport protocol for the SOAP message.
  - `style`: can have value `rpc` or `document.`
- SOAP `operation` extensibility element: provides operation scope info, has two attributes:
  - `style`: may overwrite binding-wide sittings for individual operations.
  - `soapAction`: specifies the value of the `soapAction` HTTP header for the operation.

SOAP extensibility elements

- SOAP `body` extensibility element: defines the mapping of abstract message parts into the SOAP message `body`:
  - `parts`: contains the set of abstract message parts in message `body`.
  - `use`: can take value `literal` or `encoded`.
  - `encodingStyle`: if `encoded`, this defined the encoding rules.
  - `namespace`: holds the namespace of the operation name.

SOAP RPC style binding using Section 5 encoded message parts

- `getPhoneNumber` operation, or procedure call, is identified using namespace attribute.

```
<operation name="getPhoneNumber"
  <SOAP-EXT:operation soapAction=""/>
  <input>
    <SOAP-EXT:body use="encoded"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
     namespace="http://companyx.com/ns/employees"/>
  </input>
  <output>
    <SOAP-EXT:body use="encoded"
     encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
     namespace="http://companyx.com/ns/employees"/>
  </output>
</operation>
```

SOAP address extensibility element for SOAP over HTTP, example

- This extensibility element specifies the network address of the SOAP-accessible service.
- It is a member of the WSDL `port` element.
- It contains one attribute:
  - `location`: defines the network address for the binding.

```
<service name=“PhoneNumberService”>
  <port name=“PhoneNumberPort” binding=“tns:PhoneNumberSoapBinding”>
    <SOAP-EXT:address
     location=“http://www.companyx.com/servlet/rpcrouter”/>
  </port>
</service>
```

Practices 1

- Create a web application to consume a web service called (GlobalWeather) available on the web:

  http://www.webservicex.net/globalweather.asmx
- Use the WSDL file of the web service available on:

  http://www.webservicex.net/globalweather.asmx
- Create new ASP.Net web site
- In the design add a button and a text box.
- To the solution name add a web reference (Use the link of the WSDL file above)
- Go to the design, double click on the button. Add: using (the referenced webservice name).

Button Click code:

```
protected void Button1_Click(object sender, EventArgs e)
    {
        GlobalWeather G = new GlobalWeather();
        string cities = G.GetCitiesByCountry(TxtCountry.Text.ToString());
        Response.Write(cities);
    }
```

Practice 2: Self Hosting Service

- Create a service library (any service), name your solution (SelfHostingService)
- Add a new console application project to the solution, (MyserviceHost)
  - Here, add references, to your service library and to .Net System.ServiceModel
  - Add inside the main class ,the following code:

```
ServiceHost host = new ServiceHost(typeof(MyService));
host.Open(); //read service configurations from .config file and build
    the runtime to support all endpoints defined in the configuration
Console.WriteLine("MyService is up and running ...");
Console.ReadLine();
host.Close();
```

  - Build your solution
  - Add an item (application configuration file) for the project and edit it using WCF editor:
    - Create new service (browse for your the service library), then next. Use http binding , next: use basic WS interoperability, nex: address: use relative address (e.g. name it basic) and finish.
    - Create another service endpoint, the same, but use advanced WS interoperability, and use relative address (ws)
    - Click Host in the configuration and create new base address (http://localhost:8080/MyService)
    - Create another service endpoint, use tcp binding, use address (http://localhost:8080/MyService)
    - Create another service endpoint, use Named pipes binding, use address (net.pipe://localhost/MyService)
    - To enable metadata in your service; click advanced, then service behaviour, and new service behaviour, and add the service meta data element, then select the element from the tree and set httpGetEnabeled to true
    - Click the service name in the top, and Apply the new behaviour to the behaviour configuration
    - Save the configuration and exit.

Practice 2: Self Hosting Service

- Right-click your host project and set as startup project.
- Press Crtl+F5 to start.
- Open internet explorer and in the address (localhost:8080/MyService)

Summary

During this session we have introduced the use of WSDL to describe web services; Thus, the following subjects have been described:

1. WSDL building blocks
2. Structure of a WSDL document
3. The SOAP binding

In the next session we will cover the enterprise service bus “ESB”

References

1. Aaron Skonnard, Understanding WSDL, Microsoft Digital Network, “http://msdn.microsoft.com/en-us/library/ms995800.aspx”, March 2003
2. Extracted from: http://en.wikipedia.org/wiki/WSDL
3. Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, “Perspectives on Web services- Applying SOAP, WSDL and UDDI to real-world projects, 2nd edition, Springer, 2005

Thanks

Mohammed Aldasht