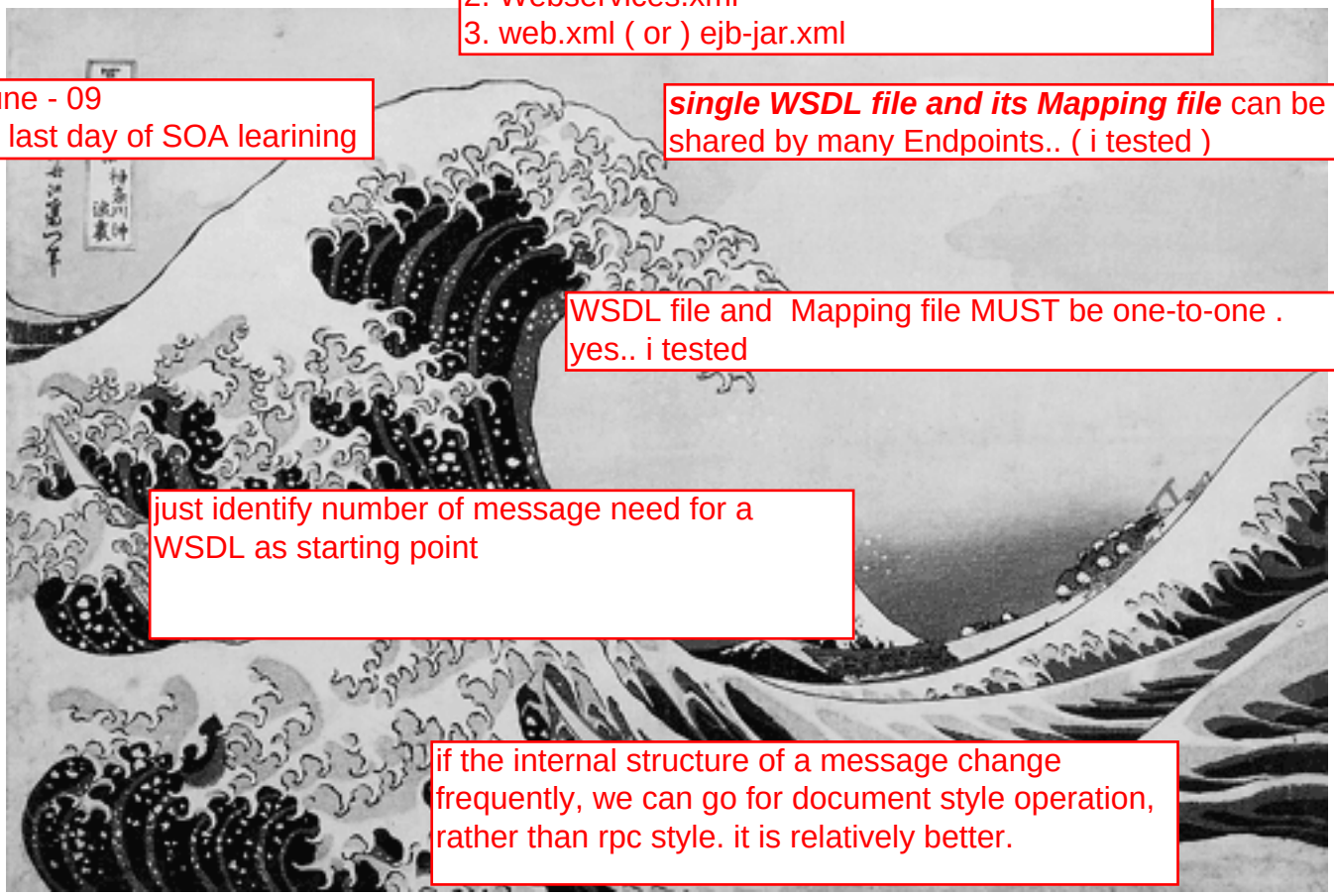
The following file must be in same ARCHIEVE. (i.e) same WAR (JSE) or same JAR (EJB)
..
1. WSDL
2. jax-rpc mapping file
2. Webservices.xml
3. web.xml (or) ejb-jar.xml

Part II: SOAP and WSD

15 - june - 09
is the last day of SOA learining

single WSDL file and its Mapping file can be shared by many Endpoints.. (i tested)



WSDL file and Mapping file MUST be one-to-one . yes.. i tested

just identify number of message need for a WSDL as starting point

if the internal structure of a message change frequently, we can go for document style operation, rather than rpc style. it is relatively better.

~~At the heart of Web services today are SOAP and WSDL, so it's important that you have a good understanding of them and how they're used. That said, memorizing the details of SOAP and WSDL is not critical. While these technologies are central to Web services, in many cases you may not deal with them directly, as they will be hidden in the communication and deployment layer of the J2EE Web Services platform.~~

~~This part of the book covers SOAP~~ ~~ort for SOAP 1.1 and WSDL 1.1 is required by the WS-I~~ ~~P Messages with Attachments is not. SwA is a sign~~ ~~tice today, however, and it's supported by J2EE Web Servic~~ ~~he BP, version 1.1, so it's covered in Appendix E.~~

but, for company project RPC style is best, since, it is tightly controled .
if you develop any tool, might consider about document style operation

~~Once you have read Part II, you should have a pretty decent understanding of SOAP 1.1 and WSDL 1.1. If you desire more detailed knowledge, I suggest you read the Notes describing these technologies, published by the World Wide Web Consortium. You must complement that reading with study of the Basic Profile, however, because the BP imposes lots of restrictions and provides many clarifications that make SOAP 1.1 more interoperable and WSDL 1.1 more portable. Still, for most developers the level of coverage in this part of the book will be more than sufficient.~~

In This Part

Related Appendices

34 pages.

current version 1.1

Chapter 5. WSDL

~~To use SOAP with a particular Web service, you need to know in advance how the SOAP messages are structured, which protocol will be employed (HTTP or SMTP, for example), and the Internet address of the Web service. In a word, you need *documentation*. For example, the BookQuote SOAP message is pretty simple, but how did you learn about it? You read about it in the preceding chapter; it's documented in this book. Listing 5-1 shows the BookQuote SOAP 1.1 request message.~~

Listing 5-1 A BookQuote RPC/Literal SOAP Request Message

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote">
   <soap:Body>
      <mh:getBookPrice>
          <isbn>0321146182</isbn>
      </mh:getBookPrice>
   </soap:Body>
</soap:Envelope>
```

~~While the description of the BookQuote Web service presented in Chapter 4 is easy to understand, it's not well documented. Imagine that all Web services were described this way. You would have to read an informal document—a chapter in a book, a Web page, or an owner's manual perhaps—every time you wanted to use a new Web service. To avoid problems associated with informal documentation, the Web services community has adopted the **Web Services Description Language (WSDL),** which is a document format for precisely describing Web services.~~

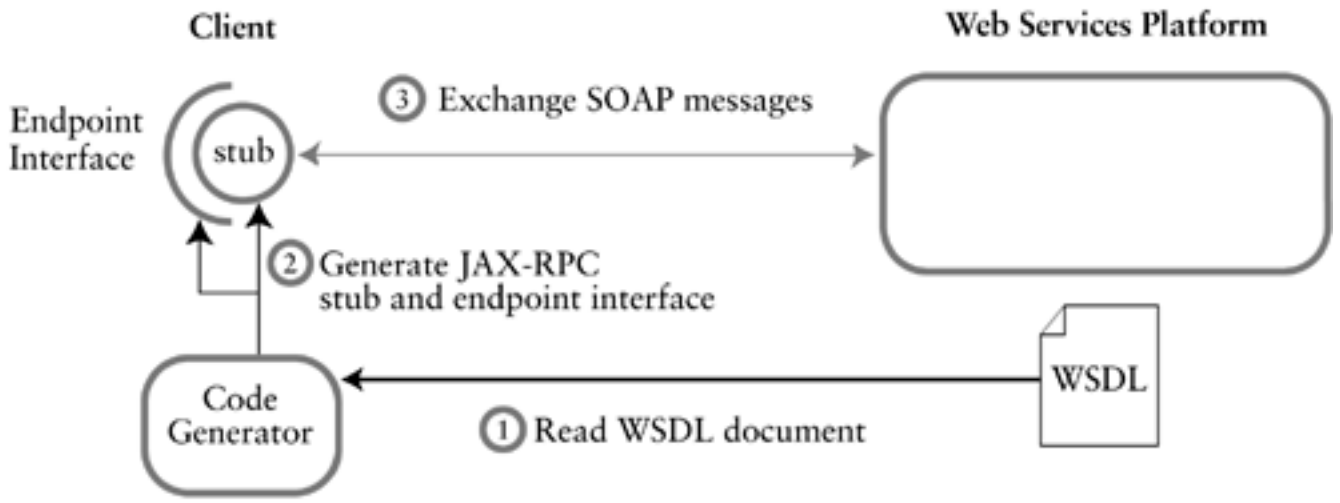
~~WSDL (routinely pronounced "whiz-dul") is used to specify the exact message format, Internet protocol, and address that a client must use to communicate with a particular Web service. In other words, a WSDL document tells us how to use a Web service. WSDL is another de facto standard for Web services and, like SOAP, it enjoys widespread adoption; it has been endorsed by most enterprise software vendors and major standard organizations, including W3C, WS-I, and OASIS.~~

~~WSDL is well suited for code generators, which can read a WSDL document and generate a programmatic interface for accessing a Web service. For example, a JAX-RPC provider uses WSDL 1.1 to generate Java RMI interfaces and network stubs, which can be used to exchange SOAP messages with a Web service.~~

> ~~***A JAX-RPC provider is a vendor implementation of the JAX-RPC API. For example, BEA WebLogic is a JAX-RPC provider. All J2EE 1.4 application servers are JAX-RPC providers, because they all provide their own implementations of the JAX-RPC API.***~~

~~JAX-RPC is not the only technology that can generate interfaces and network stubs from WSDL documents. There are many other code generators, including tools in IBM WebSphere, Microsoft .NET, and Apache Axis, to name a few. Figure 5-1 illustrates how a JAX-RPC toolkit would use a WSDL document to generate a Java RMI interface (an **endpoint interface**) and a networking stub that implements that interface.~~

Figure 5-1. A WSDL Code Generator



~~While WSDL is especially useful for code generators, it's also an asset when using other Web services tools and APIs. Many Web service clients, for example, use SOAP APIs instead of generated call interfaces and stubs. These APIs usually model the structure of the SOAP message using objects like `Envelope`, `Header`, `Body`, and `Fault`. Examples of SOAP APIs include SAAJ (SOAP with Attachments API for Java), Perl::Lite, Apache SOAP, and others. When you use a SOAP API, you can use a Web service's WSDL as a guide for exchanging SOAP messages with that Web service.~~

~~Although WSDL is considered a Web services standard,[1] it's not very simple, which is perhaps its greatest handicap. Its complexity results from its designers' intention to create an IDL (interface definition language) for Web services that is not tied to any specific protocol, programming language, or operating system.~~ Note that WSDL 1.1 is not specific to SOAP; it can be used to describe non-SOAP-based Web services as well.

> ~~[1] WSDL 1.1 was submitted as a "note" to the W3C by IBM and Microsoft in March of 2001. The next version of WSDL, WSDL 1.2, is being developed under the auspices of the W3C and will become a full recommendation.~~

~~Modularity was another design goal. Because WSDL is very modular, you can reuse its artifacts to describe more than one Web service. Unfortunately, modularity makes WSDL documents difficult to understand at first. This chapter's purpose is to enable you to make sense of WSDL so that you can write WSDL documents that describe your Web services, and read WSDL documents that describe others' Web services. That said, in many cases you may never need to read or write WSDL documents yourself because your J2EE platform will usually create and process them automatically.~~

5.1 The Basic Structure of WSDL

~~A WSDL document is an XML document that adheres to the WSDL XML schema. As an XML document instance, a WSDL document must use the correct elements in the correct fashion if it is to be valid and well formed. The rest of this chapter explains the structure of a WSDL document and how to construct one that describes a Web service properly. While WSDL can, in theory, be used to describe any kind of Web service, this chapter focuses on WSDL documents that describe Web services that are SOAP-based and compliant with the WS-I Basic Profile 1.0 (BP).~~

~~A WSDL document contains seven important elements: `types`, `import`, `message`, `portType`, `operations`, `binding`, and `service`, which are nested in the `definitions` element, the root element of a WSDL document. Figure 5-2 illustrates the basic structure of a WSDL document.~~

Figure 5-2. The Basic Structure of a W

it can be any arbitrary name. this attribute no need

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
    <xsd:schema
    targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuot
     <!-- The ISBN simple type -->
     <xsd:simpleType name="ISBN">
       <xsd:restriction base="xsd:string">
         <xsd:pattern value="[0-9]{9}[0-9Xx]" />
       </xsd:restriction>
     </xsd:simpleType>

   </xsd:schema>
 </types>
   <message name="GetBookPriceRequest">
   <part name="isbn" type="mh:ISBN" />
 </message>
 <message name="GetBookPriceResponse">
   <part name="price" type="xsd:float" />
 </message>
   <portType name="BookQuote">
   <operation name="getBookPrice">
     <input name="isbn" message="mh:GetBookPriceRequest"/>
     <output name="price" message="mh:GetBookPriceResponse"/>
   </operation>
 </portType>
   <binding name="BookQuote_Binding" type="mh:BookQuote">
   <soapbind:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="getBookPrice">
     <soapbind:operation style="rpc"
      soapAction=
      "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
      <input>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </input>
```

i have to go for <types> if Only if, i need either **simpleType** or **complexType.** otherwise i never need use this element

if i am not using any simpleType or complexType and not <types> element used, i have to declare namespace of xmlSchema here. .... otherwise just declare in <schema> element of <types>.

this <soap:operation> is optional for "document" styled operation and mandatory for "rpc" styled operation. (i tested)

```
      <soapbind:body use="literal"
       namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </input>
    <output>
      <soapbind:body use="literal"
       namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </output>
  </operation>
</binding>

<service name="BookQuoteService">
  <port name="BookQuote_Port" binding="mh:BookQuote_Binding">
    <soapbind:address location=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote" />
  </port>
</service>

</definitions>
```

~~**The types element** uses the XML schema language to declare complex data types and elements that are used elsewhere in the WSDL document.~~

~~**The import element** is similar to an import element in an XML schema document; it's used to import WSDL definitions from other WSDL documents.~~

~~**The message element** describes the message's payload using XML schema built-in types, complex types, or elements that are defined in the WSDL document's types element, or defined in an external WSDL document the import element refers to.~~

~~**The portType and operation elements** describe a Web service's interface and define its methods. A portType and its operation elements are analogous to a Java interface and its method declarations. An operation element uses one or more message types to define its input and output payloads.~~

~~**The binding element** assigns a portType and its operation elements to a particular protocol (for instance, SOAP 1.1) and encoding style.~~

~~**The service element** is responsible for assigning an Internet address to a specific binding.~~

~~**The documentation element** explains some aspect of the WSDL document to human readers. Any of the other WSDL elements may contain documentation elements. The documentation element is not critical, so it will not be mentioned again in this chapter.~~

~~Listing 5-2 is an example of a very simple WSDL document, which describes the BookQuote Web service discussed in Chapter 4. When you look at the WSDL definition, keep in mind that it's supposed to describe a Web service and the types of SOAP messages, protocols, and Internet addresses used to access that Web service.~~

Listing 5-2 The WSDL Definition for the BookQuote Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
```

```
xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- message elements describe the input and output pa
  <message name="GetBookPriceRequest">
    <part name="isbn" type="xsd:string" />
  </message>
  <message name="GetBookPriceResponse">
    <part name="price" type="xsd:float" />
  </message>

  <!-- portType element describes the abstract interface of a We
  <portType name="BookQuote">
    <operation name="getBookPrice">
      <input name="isbn" message="mh:GetBookPriceRequest"/>
      <output name="price" message="mh:GetBookPriceResponse"/>
    </operation>
  </portType>

  <!-- binding tells us which protocols and encoding styles are used -->
  <binding name="BookPrice_Binding" type="mh:BookQuote">
    <soapbind:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getBookPrice">
      <soapbind:operation style="rpc"
       soapAction=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
        <input>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </input>
        <output>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </output>
    </operation>
  </binding>

  <!-- service tells us the Internet address of a Web service -->
  <service name="BookPriceService">
    <port name="BookPrice_Port" binding="mh:BookPrice_Binding">
      <soapbind:address location=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </port>
  </service>

</definitions>
```

message can be empty. for example, a method take nothing as input parameter. even we need a empty message just because of to invoke.

<message> element just act as WRAPPER for one or more schema components. nothing else

message used for <input> and <output> can be empty. but <fault> <soap:header>, <soap; headerfault> msg cannot be empty. (i tested)

<input> is mandatory for every operation

this <soap:operation> is optional for "document" styled operation and mandatory for "rpc" styled operation. (i tested)

~~You're not expected to understand this document at this point. If you're confused by its structure, don't despair—that's the normal reaction for developers just learning WSDL. The rest of this chapter explains in detail how a WSDL document is organized and what all its elements and attributes do—and explains Listing 5-2 piece by piece.~~

5.2 WSDL Declarations: The `definitions`, `types`, and `import` Elements

~~This section focuses on the first two child elements of the `definitions` element (the root element), `types` and `import`, which define the data types and other artifacts used by the WSDL document. Before we examine these crucial elements, though, we need to look at the beginning of the document.~~

5.2.1 The XML Declaration

The XML declaration in Listing 5-2 specified a character encoding of UTF-8.

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

A WSDL document must use either UTF-8 or UTF-16 encoding; other encoding systems are not allowed.[BP]

5.2.2 The `definitions` Element

~~The root element of all WSDL documents is the `definitions` element, which encapsulates the entire document and also provides a WSDL document with its name.~~

```
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
```

~~The `definitions` element usually contains several XML namespace declarations, which is normal for a root element. Among these declarations is one for the namespace of the WSDL 1.1 XML schema "http://schemas.xmlsoap.org/wsdl/". Declaring the WSDL namespace as the default namespace avoids having to qualify every WSDL element explicitly, with a prefix.~~

The first attribute you see in the `definitions` element is `name`, which is used to name the entire WSDL document. In practice the WSDL name is not all that important. Nothing refers to it, and it's optional.

The `definitions` element also declares a `targetNamespace` attribute, which identifies the namespace of elements defined in the WSDL document—much as it does in XML schema documents.

~~The `message`, `portType`, and `binding` elements are assigned labels using their `name` attributes; these labels automatically take on the namespace specified by the `targetNamespace` attribute. (In this book the URL of the `targetNamespace` is also assigned a prefix of `mh`.) Labeled `message`, `portType`, and `binding` elements are commonly called **definitions.** These definitions assume the namespace specified by `targetNamespace`.~~

~~Other elements in the document refer to the definitions using their label and namespace prefix. A prefixed label is considered a fully qualified name (QName) for a definition. For example, in the following snippet~~

~~from Listing 5-2, input and output elements refer to the message definitions using their QNames.~~

```
<!-- message elements describe the input and output parameters -->
<message name="GetBookPriceRequest">
  <part name="isbn" type="xsd:string" />
</message>
<message name="GetBookPriceResponse">
  <part name="price" type="xsd:float" />
</message>

<!-- portType element describes the abstract interface of a Web service -->
<portType name="BookQuote">
  <operation name="getBookPrice">
    <input name="isbn" message="mh:GetBookPriceRequest"/>
    <output name="price" message="mh:GetBookPriceResponse"/>
  </operation>
</portType>
```

5.2.3 The `types` Element

~~WSDL adopts, as its basic type sy~~ ~~s as a~~
~~container for defining any data ty~~ ~~k~~
~~types and custom simple types. T~~ ~~y~~
~~message definitions when declarin~~ ~~g the parts (payloads) of messages.~~

<types> used to define new schema component,
...
where as <import> of WSDL is used to reuse of exiting WSDL'***s*** schema components

~~The `types` element is not used in Listing 5-2, because it's unnecessary—the message definitions, `GetBookPriceRequest` and `GetBookPriceResponse`, refer to simple built-in types.~~

~~If we wanted to, we could define a custom simple type for ISBN number and use that in the `GetBookPriceRequest` message instead of the built-in `string` type, as in Listing 5-3.~~

~~Listing~~ 5-3 Using XML Schema Types Defined in the WSDL `types` Element

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/
 xmlns:soapbind="http://schemas.xmlsoap.org/wsd
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>

    <xsd:schema
     targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote">
      <!-- The ISBN simple type -->
      <xsd:simpleType name="ISBN">
        <xsd:restriction base="xsd:string">
          <xsd:pattern value="[0-9]{9}[0-9Xx]" />
        </xsd:restriction>
      </xsd:simpleType>
```

just declare this name space in <schema> instead of <definitions> element if <schema> exist in wsdl file

this "targetNamespace" in Schema is must

we have to declare <schema> only if we need any simpleType or complexType. for just use of built-in type of XMLSchema no this <types> element in WSDL

```
    </xsd:schema>

  </types>

  <!-- message elements describe the input and output parameters -->
  <message name="GetBookPriceRequest">
    <part name="isbn" type="mh:ISBN" />
  </message>
  <message name="GetBookPriceResponse">
    <part name="price" type="xsd:float" />
  </message>
  ...
</definitions>
```

~~In Listing 5-3 a complete W3C XML schema document is nested directly in the `types` element. The custom simple type, labeled `ISBN`, is defined and assigned to the `mh` namespace by the XML schema `targetNamespace` attribute. The `mh:ISBN` type is then used to define the `isbn` part of the `GetBookPriceRequest` message definition.~~

~~The `targetNamespace` attribute of the XML schema must be a valid non-null value, otherwise the types and element will not belong to a valid namespace. In addition, the XML schema defined in the `types` element must belong to a namespace specified by the WSDL document (usually in the `definitions` element) or to a namespace of an imported WSDL document.[BP] In other words, the WSDL document must be aware of any and all namespaces used in the document. The mechanism for importing WSDL documents is discussed in the next section.~~

SOAP and WSDL both define attributes and encoding types you can use to define array data types. Such array types and attributes have created a lot of confusion and interoperability problems, however, so the BP strictly prohibits their use. In a nutshell, you are not allowed to use the `Array` type, or the `arrayType` attribute defined for SOAP 1.1 Encoding (SOAP 1.1 Note, Section 5), or the WSDL `arrayType` attribute defined by WSDL. In addition you should not label array types as "ArrayOfXXX" as suggested by the WSDL 1.1 Note.[BP] These requirements are not a handicap at all, because XML schema provides a much simpler way to define arrays. All you need to do is define a complex type with a `maxOccurs` value greater than 0 (for example, `"10"`, `"32000"`, or `"unbounded"`) and you have yourself a basic array. The following snippet defines an array-like type in this way.

according to BP, arrayType of SOAP / WSDL should not use.

But, we can achieve the same by maxOccurs value.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>

    <xsd:schema
     targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote">
      <!--A simple array-like type -->
      <xsd:complexType name="IntArray">
        <xsd:sequence>
```

```
        <xsd:element name="arg" type="xsd:int" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

  </xsd:schema>
</types>
```

the imported file is not found, the deployment time itself we get exception. i tested

<import> element of WSDL can be used to import either another WSDL or XSD, even it is designed to import WSDL only. i tested

5.2.4 The `import` Element

the best situation to import another WSDL file is, many portType sharing same Fault message

~~The `impor` ... nt the definitions from a specified namespace ... if you want to modularize WSDL documents—for example, to separate the abstract definitions (the `types`, `message`, and `portType` elements) from the concrete definitions (the `binding`, `service`, and `port` elements). Another reason to use `import` ... nt to maintain separately. For exampl ... ping and order processing separat ... irectory accessible by business partner~~

<types> used to define new schema component,
...
where as <import> of WSDL is used to reuse of exiting WSDL's schema components

```
<definitions name="AllMhWebServices"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

    <import namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
     location="http://www.Monson-Haefel.com/jwsbook/BookPrice.wsdl"/>
    <import namespace="http://www.Monson-Haefel.com/jwsbook/po"
     location="http://www.Monson-Haefel.com/jwsbook/wsdl/PurchaseOrder.wsdl"/>
    <import namespace="http://www.Monson-Haefel.com/jwsbook/Shipping"
     location="http://www.Monson-Haefel.com/jwsbook/wsdl/Shipping.wsdl"/>

</definitions >
```

'location' cannot be empty

so, "location" attribute can point anywhere even outside of current archieve ... this no need to be a WSDL . it can be XSD too.. i tested

The WSDL `import` element must declare two attributes: `namespace` and `location`. The value of the `namespace` attribute must match the `targetNamespace` declared by the WSDL document being imported. The `location` attribute must point to an actual WSDL document; it cannot be empty or null. ~~That said, the `location` is considered a hint. If the application reading the WSDL document has cached or stored copies of the imported WSDL document locally, it may use those instead.~~[BP]

> ~~***Use of the `import` element is convenient, but it can also create versioning headaches. If WSDL documents are maintained separately, the risk of an imported document being changed without regard to the WSDL documents that import it is pretty high. Take care to ensure that imported WSDL documents are not changed without considering versioning.***~~

You can use `import` and `types` together, but you should list the `import` elements before the `types` element in a WSDL document.

it same as usual in java, import shoud be first and then actual new class definition

~~There has been a lot of confusion about the purpose of the WSDL `im` ... e believed it could be used to import either WSDL documents or XML schema documen ... eved it was only for importing WSDL documents. Varying interpretations have created i ... ems, so the Basic Profile specifies that a WSDL `import` element may refer only to WS ... u need to import an XML schema element, you should do so in the XML schema definitio ... VSDL `types`~~

~~element, using the standard XML schema `import` statement as described in Section 3.2.5.~~ It's important to note that you cannot use the XML schema `import` statement to import an XML schema directly from the `types` element of some other WSDL document.[BP]

[Team LiB

oh.. oh...
..
In real life, create xml schema separately and import that into WSDL. it is helpful to write JAXB coding for client writing for a web service
...
fortunately, we can import either XSD or WSDL file. it is better import a XSD in <types> element

it is peferable, importing a XSD file, instead of importing a WSDL file.

so, while importing, we can use either <import> of wsdl or <import> of schema.
..
But we cannot directly import schema which is local to another WSDL, so we get those localy scoped schema only through import its WSDL document as whole.
..
i would suggest, define schema separately and import in any WSDL file rather than trying to reuse whole WSDL file

the Imported schema can have different namespace than targetNamespace of <definitions> in wsdl. actually it is good parctice (i tested)
....
But, value of namespace must match targetNamespace of schema while being imported..

the aim of this import, to bring schema types and element which is defined in another wsdl to current wsdl as reuse.
.....
but is does not designed to reuse other element of wsdl (my assumption)

5.3 The WSDL Abstract Interface: The `message`, `portType`, and `operation` Elements

The `message`, `portType`, and `operation` elements describe the abstract interface of the Web service. The `portType` combines the `operation` and `message` definitions into an abstract interface that is analogous to a Java interface definition. The `portType` describes the kinds of operations that a Web service supports—the messaging mode and payloads—without specifying the Internet protocol or address used.

5.3.1 The `message` Element

<message> element act as Wrapper for one or more XMLSchema components via <part> element

The `message` element describes the payload of a message used by a Web service. A `message` element can describe the payloads of outgoing or incoming messages—that is, messages that are directly sent to or received from a Web service. In addition, the message element can describe the contents of SOAP header blocks and fault `detail` elements. The way to define a `message` element depends on whether you use RPC-style or document-style messaging.

5.3.1.1 The `message` Element for RPC-Style Web Services

Message used in 5 palces
.....
1. <input>
2. <output>
3. <fault>
4. <soap:header>
5. <soap:headerfault>
.
Message can two types
.....
1. RPC styled operations
2. Document styled operations

so, style of message is used just who will be immediate child of soap BODY either name of operation or schema component

When RPC-sty elements describe the pa d reply messages. Th , call return values, hea ble, the BookQuote W ments to describe the p hat service. (Mes er blocks are discussed 5-2 illustrates.

<!— message elements describe the input and output paramet

As a good practice, ALWAYS define <output> message.
....
AT LEAST empty message even if java method has 'void' as return.
...
This will help to add fault at some time later.

The `GetBookPriceRequest` message represents the parameters (the input), while `GetBookPriceResponse` represents the reply (the output). In other words, the `GetBookPriceRequest` message definition describes the payload of the message transmitted from a SOAP client to the BookQuote Web service, while the `GetBookPriceResponse` message definition describes the payload of the message transmitted by the BookQuote Web service back to the client.

There is no prescribed convention for naming messages. In this book, messages transmitted from the SOAP client to the server are suffixed with `Request`, and messages transmitted back to the client are suffixed with `Response`. You aren't required to use this convention; use any naming system you like. Message names are arbitrary and only serve to qualify a message definition. A message element cannot declare itself to be input or output—that distinction can be made only by the `operation` elements discussed in the next section—so naming a message `Request` or `Output` or whatever won't determine how it's used.

In RPC-style messaging, messages commonly have more than one part. For example, you can define a

~~message called~~ ~~GetBulkBookPriceRequest~~ ~~with mu~~
~~different parameter.~~

NAME of of <message>, <part> <portType> <binding> and <service> are just Label. cannot used as PAYLOAD. hence these are just label used for further reference in same WSDL file.
.....
name <operation> may be in payload if it is RPC style and not in payload if it is Document style message
.....
Element or Types are defined in SCHEMA are actual Payload. that is elements or types goes inside <BODY> of SOAP message
..
So, in short, Only Schema components goes inside of <body> of soap.

```
<definitions name="BookPrice" ...>
  ...
  <message name="GetBulkBookPriceRequest">
    <part name="isbn" type="xsd:string"/>
    <part name="quantity" type="xsd:int"/>
  </message>
  <message name="GetBulkBookPriceResponse">
    <part name="price" type="mh:prices" />
  </message>
  ...
</definitions>
```

so, All the <part> of ONE message can have either "type" or "element". but cannot be mixed of both, since a message can be used for RPC style or Document Style not for both purpose. but it can be mixed...i tested ... OHHH

~~Both input~~ ~~arts, which is a departure from Java~~
~~method-ca~~ ~~but only one output (the return~~
~~value). It's~~ ~~C#, and Perl to declare parameters~~
~~that are either input or output arguments. WSDL is intended to be programming-language neutral, so it~~
~~must be flexible enough to accommodate all programming languages, not just Java. Both SAAJ and JAX-~~
~~RPC, cover~~ ~~output messages with multiple parts.~~

part will be one-to-one mapping with parameter of a java method

5.3.1.2 The `message` Element for Document-Style Web Services

~~When you use document-s~~ ~~inition refers to a top-level element in the types~~
~~definition.BP For example,~~ ~~that describes a SubmitPurchaseOrder Web~~
~~Service. The Purchase Ord~~ ~~his book is imported into the types element of~~
~~the WSDL document. In th~~ ~~e-Way messaging, so there is no reply message,~~
~~which is common in docum~~

so, style of message is used just who will be immediate child of soap BODY either name of operation or schema component

Listing 5-4 Using the XML Schema `import` Element

so, while importing, we can use either <import> of wsdl or <import> of schema. But this is good practice

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PurchaseOrderWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/PO"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/PO"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <xsd:schema targetNamespace="http://www.Monson-Haefel.com/jwsbook/PO">
      <!-- Import the PurchaseOrder XML schema document -->
      <xsd:import namespace="http://www.Monson-Haefel.com/jwsbook/PO"
       schemaLocation="http://www.Monson-Haefel.com/jwsbook/po.xsd" />
    </xsd:schema>
  </types>
  <!-- message elements describe the input and output parameters -->
  <message name="SubmitPurchaseOrderMessage">
    <part name="order" element="mh:purchas
  </message>
  ...
```

so, All the <part> of ONE message can have either "type" or "element". but cannot be mixed of both, since a message can be used for RPC style or Document Style not for both purpose.. But it can be mixed. i tested. OHHHH.

```
</definitions>
```

A message part may declare either a `type` attribute or an `element` attribute, but not both. Which to use depends on the kind of messaging you're doing. If you're using RPC-style messaging, the `part` elements must use the `type` attribute; if you're using document-style messaging, the `part` elements must use the `element` attribute.BP RPC-style messaging uses types to define procedure calls, where each element represents a type of parameter. Document-style messaging, on the other hand, exchanges XML document fragments and refers to their top-level (global) elements.

5.3.1.3 Declaring Fault Messages

~~You can use message definitions to declare faults in the same way you use them to declare input and output messages. For example, Listing 5-5 defines a fault message that is sent back if a BookQuote request message contains an invalid ISBN. It includes a single part, an error message.~~

Listing 5-5 Declaring a Fault Message

```
<definitions name="BookQuote" ...>

  <types>
    <xsd:schema targetNamespace="http://www.Monson-Haefel.com/jwsbook/PO">
      <!-- Import the PurchaseOrder XML schema document -->
      <xsd:element name="InvalidIsbnFaultDetail" >
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="offending-value" type="xsd:string"/>
            <xsd:element name="conformance-rules" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </types>

  <!-- message elements describe the input and output parameters -->
  <message name="GetBookPriceRequest">
    <part name="isbn" type="xsd:string" />
  </message>
  <message name="GetBookPriceResponse">
    <part name="price" type="xsd:float" />
  </message>
  <message name="InvalidArgumentFault">
    <part name="error_message" element="mh:InvalidIsbnFaultDetail" />
  </message>
</definitions>
```

fault message cannot empty and also, i can have only one <part>

this element has to match regarding type in exception class constructor parameters

so, fault message's part can have either "type" or element.

so, a java bean parameter will be in constructor of exception class

Fault messages used by SOAP-based Web services can have only one part because, as you learned in Chapter 4, SOAP faults must adhere to a specified schema. In the case of a SOAP fault, the `part` definition refers to the contents of the detail section of the fault message. ~~While there is no industry convention for naming fault messages, it's the convention of this book to suffix the name with the word `Fault`~~.

The message definitions used by faults use Document/Literal encoding style and therefore must be based

~~on a top-level element defined in the `types` element or imported in a WSDL or XML schema document.[BP] In Listing 5-5 the `types` element defines the `InvalidIsbnFaultDetail` top-level element as an anonymous type (see Section 3.2.4 for details about anonymous types). A SOAP fault message that adhered to this definition would look something like the SOAP message in Listing 5-6.~~

Listing 5-6 A SOAP Message That Conforms to the WSDL Document Definition in Listing 5-5

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote" >
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Sender</faultcode>
      <faultstring>
        The ISBN value contains invalid characters
      </faultstring>
      <faultactor>http://www.xyzcorp.com</faultactor>
      <detail>
        <mh:InvalidIsbnFaultDetail>
          <offending-value>19318224-D</offending-value>
          <conformance-rules>
            The first nine characters must be digits. The last
            character may be a digit or the letter 'X'. Case is
            not important.
          </conformance-rules>
        </mh:InvalidIsbnFaultDetail>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

~~Message definitions can also be used to describe SOAP header blocks and fault header blocks. This SOAP-specific feature is discussed at the end of this chapter.~~

5.3.2 The `portType` Element

single portType can have BOTH "rpc" and "document" styled operation even it is not allowed by BP-1. (i tested)

~~A `portType` defines the abstract interface of a Web service. Conceptually, it's a lot like a Java interface because it defines an abstract type and its methods, but not an implementation. In WSDL the `portType` is implemented by the `binding` and `service` elements, which dictate the Internet protocols, encoding schemes, and an Internet address used by a Web service implementation. The "methods" of the `portType` are its `operation` elements. A `portType` may have one or more `operation` elements, each of which defines an RPC- or document-style Web service method. Each `operation` is composed of at most one `input` or `output` e ... "BookQuote", (shown in ... ce, and `GetBook` ... shown in the right colu ...~~

single <portType> can be shared by more than one <binding> (i tested)
..
single <binding> can be shared by more than one <port> of <service> (i tested)
....
a <service> element in WSDL, can have more than one <port> element
...
in wsdl can be more than <service> element. but usually no need

Table 5-1. Comparing a WSDL `portType` Definition to a Java Interface Definition

WSDL `portType`	Java Interface
`<portType name="BookQuote">`	`public interface BookQuote {`
`<operation name="GetBookPrice">`	`public float getBookPrice (String isbn);`
`<input name="isbn" message="mh:GetBookPriceRequest"/>`	
`<output name="price" message="mh:GetBookPriceResponse"/>`	
`</operation>`	
`<operation name="GetBulkBookPrice">`	`public float getBulkBookPrice (String isbn, int quantity);`
`<input name="request" message="mh:GetBulkBookPriceRequest"/>`	
`<output name="prices" message="mh:GetBulkBookPriceResponse"/>`	
`</operation>`	
`<operation name="GetBookIsbn">`	`public String getBookIsbn (String bookTitle);`
`<input name="title" message="mh:GetBookIsbnRequest"/>`	`}`
`<output name="isbn" message="mh:GetBookIsbnResponse"/>`	
`</operation>`	
`</portType>`	

operation name is ONLY included in payload (BODY) of SOAP if it is RPC styled operation
..
only schema element goes inside of BODY of SOAP message as payload if it is DOCUMENT styled operation

~~The analogy between a WSDL `portType` and a Java interface is not perfect, but it's very close. In fact, most Java-based Web service code generators create mappings between Java interfaces and WSDL `portType` elements. They actually generate Java interfaces from WSDL `portType` elements—and can also generate WSDL `portType`, `operation`, and `message` elements from simple Java interfaces.~~

A WSDL document can have one or more `portType` elements, ~~each of which describes the abstract interface to a different Web service. For example, a WSDL Web service might define one `portType` named "BookQuote" and another named "SubmitPurchaseOrder".~~

yes. i tested. but need 2 bindings, 2 <port>, 2 - entry in mapping file, 2- entry in webservice.xml, 2 - entry in web.xml

5.3.3 The `operation` Element

an operation cannot be without input message. but message can be without any part.

if "void java method" throws any exception, that time, just declare empty <output> then declare <fault> msg to compromise wsdl standard. (i tested)

a <portType> can be without any single <operation>. Not any deployment error. But it is meaning less (i tested)

<fualt> message Must have one Part. and it cannot be empty message as <input> and <output> msg (get runtime error)

```
<portType name="BookQuote">
  <operation name="getBookPrice">
     <input name="isbn" message="mh:GetBookPriceRequest"/>
     <output name="price" message="mh:GetBookPriceResponse"/>
     <fault name="InvalidArgumentFault" message="mh:InvalidArgumentFault"/>
  </operation>
</portType>
```

it is used in binding.
More than one <fault> possible in a operation, that why "name" attribute is mandatory

one operation can have more than one <fault>.. But only one <detail> element can be in SOAP Response message, even interface method's header having more than exceptions in its throws clause, it can throw ONLY ONE EXCEPTION AT A TIME

A ... represents the ... nts the p... client. In RPC-style messaging the input is the request, and the output is the r... nt-style messaging the input is the XML document fragment sent to the Web service, a... XML ... n addition to input and output m... on ma... of which describes a different kind o... message

It can be occur only if a operation has <output> element. It cannot be one-way message

so, fault message's part can have either "type" or "element". based on message STYLE.

5.3.3.1 Parameter Order within an Operation

So, in RPC style operation only can have "parameterOrder' attribute DO:

In WSDL, when RPC-style messaging is used, it's assumed that the client uses procedure-call semantics. For example, JAX-RPC uses Java RMI interfaces with method calls to model RPC-style SOAP-based Web services. In many cases the parameters of the input and output messages must be transferred in a specific ... ers of the procedure call. To enforce proper ordering of input or outp... n element may declare a parameterOrder attribute. For example ... GetBulkBookPrice" uses the parameterOrder attribute to specify ...

if <output> message has more than one <part>, 'parameterOrder' is must to identify which is the returning part in a operation.

Listing 5-8 Using the parameterOrder Attribute

so, the "parameterOrder" is used only in RPC style message..

```
<message name="GetBulkBookPriceRequest">
  <part name="isbn" type="xsd:string"/>
  <part name="quantity" type="xsd:int"/>
</message>
<message name="GetBulkBookPriceResponse">
  <part name="prices" type="mh:prices" />
</message>
<portType name="GetBulkBookPrice" >
  <operation name="getBulkBookPrice" parameterOrder="isbn quantity">
     <input name="request" message="mh:GetBulkBookPriceRequest"/>
     <output name="prices" message="mh:GetBulkBookPriceResponse"/>
  </operation>
</portType>
```

According to BS-I, the usage of "parameterOrder" to identify which is return part when more than one part defined in output message

separated by SPACE, not comma (i tested)

When a parameterOrder attribute is used, it must include all the input parts and only the output parts that are not the return type. So if an output has only one part, as in the preceding example, it's assumed to be the return value and should not be listed by the parameterOrder attribute. If an output part is listed by the parameterOrder attribute, it's treated as an OUT parameter. When the input and output elements in a portType declare a part with the same name, it's an INOUT parameter, and the type must be the same in the input and output elements. OUT and INOUT type parameters are alien to most Java developers, but they are familiar features in C++, C#, and ... explanation of OUT and INOUT parameters and how they a...

if i missed any parts, the error message say clearly as "less parameter for method (i tested)

1. INOUT parameter (a <part> name of input and output msg) must be in same type in both input and output message
2. if parameterOrder must declare all parts except return part. that is, all <part> of input msg and all <part> of output except returning <part>

the input message definition named "GetBulkBookPrice". The purpose of the parameterOrder attribute is to indicate which part, if any, is the return type. Any part that is omitted from the list provided by the parameterOrder attribute is assumed to be the return type of the operation. A procedure call can have only one return type, so only a single output part may be omitted from the parameterOrder attribute.[BP]

5.3.3.2 Operation Overloading

WSDL supports operation overloading that is very similar to Java method overloading. In WSDL, two operations may have the same name, provided their input or output messages differ. Unfortunately, this feature has caused enough interoperability problems that the Basic Profile prohibits operation overloading. Every operation defined by a particular portType must have a unique name. That said, it's perfectly acceptable for two or more portType elements to declare operation elements with the same name, because each portType is considered a separate definition.[BP]

yes. it possible .(i tested.)

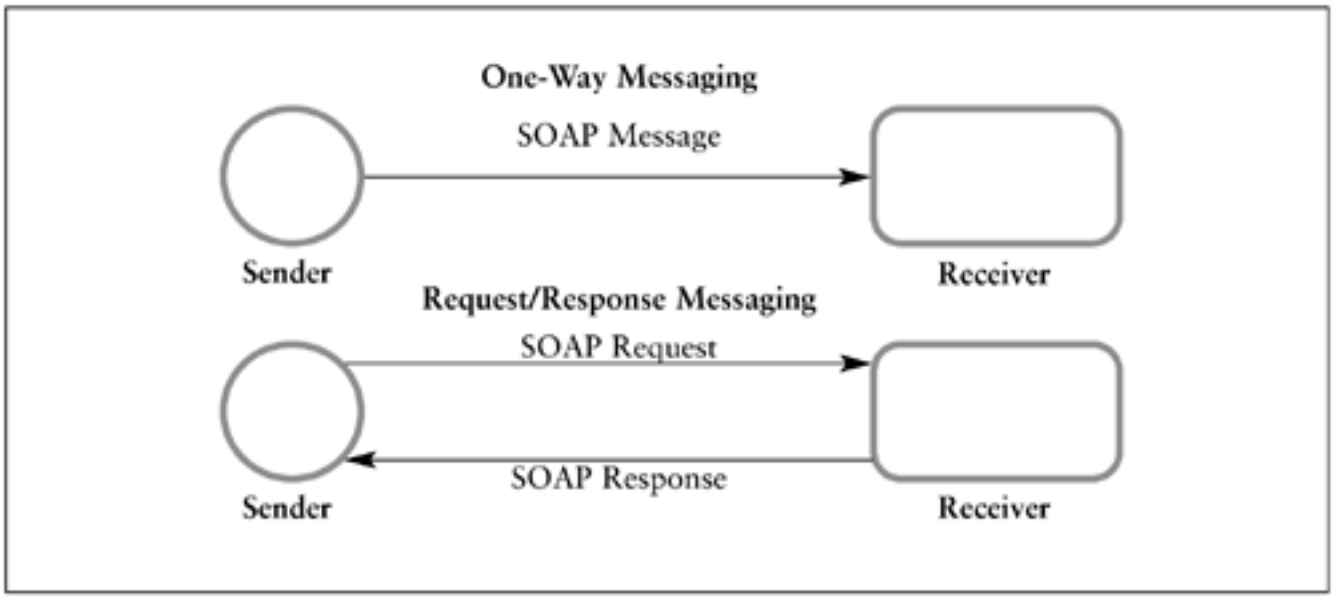
5.4 WSDL Messaging Exchange Patterns

There are four basic message exchange patterns (MEPs) used in Web services: Request/Response, One-Way, Notification, and Solicit/Response. ~~Although Notification and Solicit/Response messaging are supported by WSDL, they are not supported by the Basic Profile,BP and are rarely used in practice. Most WSDL-based Web services today use either Request/Response or One-Way messaging,~~[2] ~~and they're the only MEPs that can be used with J2EE Web Services.~~

[2] ~~This may change with the introduction of WSDL 1.2, which supports a larger set of messaging patterns.~~

~~A WSDL document can dictate the MEP for a specific operation by the way it declares its input and output elements. Figure 5-3 illustrates the chief difference between Request/Response and One-Way messaging: In the former, the sender expects a reply; in the latter it doesn't.~~

Figure 5-3. Comparing Messaging Modes: Request/Response versus One-Way



5.4.1 Request/Response Messaging

~~In Request/Response messaging the client initiates the communication by sending the Web service a request message, and the Web service replies with a response message.~~

~~If an `operation` is declared with a single `input` element followed by a single `output` element, it defines a Request/Response operation. By listing the `input` first, the `operation` indicates that the Web service receives a message that is initially sent by the client. Listing the `output` second indicates that the Web service should respond to the message. The following snippet from Listing 5-2 represents a classic Request/Response operation, with exactly one `input` and one `output`.~~

```
<!-- portType element describes the abstract interface of a Web service -->
<portType name="BookQuote">
  <operation name="getBookPrice">
    <input name="isbn" message="mh:GetBookPriceRequest"/>
    <output name="price" message="mh:GetBookPriceResponse"/>
  </operation>
</portType>
```

In addition to its one `input` and one `output`, a Request/Response operation may also include `fault` elements, which are returned to the client in the event of an error. A Request/Response operation can have zero or more faults. The following snippet illustrates.

```
<portType name="BookQuote">
  <operation name="getBookPrice">
     <input name="isbn" message="mh:GetBookPriceRequest"/>
     <output name="price" message="mh:GetBookPriceResponse"/>
     <fault name="InvalidArgumentFault" message="mh:InvalidArgumentFault"/>
     <fault name="SecurityFault" message="mh:SecurityFault"/>
  </operation>
</portType>
```

<input> MUST be one (may empty msg)
<output> can be zero or ***one***
but, <fault> can be zero or more than one

5.4.2 One-Way Messaging

~~In One-Way messaging, the client sends a message to a Web service, but doesn't expect a reply message. This MEP is typically thought of as asynchronous messaging. Next to Request/Response, it is the most popular MEP employed today.~~

~~If an operation is declared with a single `input` but *no* `output`, it defines a One-Way operation. By listing only an `input` message, the `operation` indicates that clients will send messages to the Web service without expecting a response. The following snippet shows the `SubmitPurchaseOrder_portType` that defines a One-Way operation.~~

```
<portType name="SubmitPurchaseOrder_PortType">
  <operation name="SubmitPurchaseOrder">
     <input name="order" message="mh:SubmitPurchaseOrderMessage"/>
  </operation>
</portType>
```

Unlike Request/Response operations, One-Way operations may not specify fault elements and do not generate fault messages. The messaging model is strictly unidirectional—faults cannot be sent back to the client.

so <fault> cannot be without <output>
(i tested)

~~5.4.3 Notification and Solicit/Response Messaging~~

~~Neither the Notification nor the Solicit/Response MEP can be used in J2EE Web Services. The unwillingness to support these styles is practical because they are poorly specified by the WSDL 1.1 specification and tend to introduce more problems than they solve. On the other hand, it's probably a good idea for you to understand the basic mechanics of these MEPs just for general purposes.~~

In Notification messaging the Web service sends a message to a client, but doesn't expect a reply message.

~~A Web service that uses the Notification MEP follows the **push model** of distributed computing. The assumption is that the client has registered with the Web service to receive messages (notifications) about an event. The clients that register to receive notifications are called **subscribers.** In Notification messaging, the `portType` contains an `output` element but *no* `input` message definitions.~~

~~Solicit/Response is similar to Notification, except that the client is expected to respond to the Web service. As with Notification messaging, clients of Solicit/ Response Web services must subscribe to the service in order to receive messages. In this MEP the `portType` first declares an `output` message, then an `input` message—exactly the reverse of a Request/Response operation.~~

[Team LiB]

PREVIOUS NEXT

[Team LiB]

3 - May -09

single portType can have "rpc" and "document" styled operation even it is not allowed by BP-1. (i tested)

5.5 WSDL Implementation: The `binding` Element

The `binding` element maps an abstract `portType` to a set of concrete protocols such as SOAP and HTTP, messaging styles (RPC or document), and encoding styles (Literal or SOAP Encoding). The `binding` e nts. The `binding` e the protocol-specific elements declare the protocol and encoding style to be associated with the `portType`. Each type of protocol (SOAP, MIME, and HTTP) has its own set of protocol-specific elements and its own namespace. For example, the following snippet from Listing 5-2 declares that the "BookQuote" portType is bound to the SOAP 1.1 protocol using SOAP-specific protocol elements.

a wsdl cannot be without any one binding like soap. schema validation error we would get through <port>. since <port> expect any one binding (i tested)

can single <binding> element shared by more than one <port> element?
...
ANS: yes. it can be shared for load balance purpose. (i tested)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  ...
  <!-- binding tells us which protocols and encoding styles
  <binding name="BookPrice_Binding" type="mh:BookQuote">
    <soapbind:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getBookPrice">
      <soapbind:operation style="rpc"
       soapAction=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetB
        <input>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </input>
        <output>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </output>
    </operation>
  </binding>
  ...
</definitions>
```

single portType, can have different "style" ed operations.. yes it possible in J2EE.(i tested)
..
but BS-I not allow by expecting style attribute must have same value in both <binding> and <operation>

The first thing you should notice is that the `binding` element declared in Listing 5-2 is actually composed of two different namespaces. The elements without a namespace prefix (shown in bold) are members of the WSDL 1.1 namespace "http://schemas.xmlsoap.org/wsdl/", which is the default namespace of the WSDL document. The WSDL 1.1-generic binding elements are `binding`, `operation`, `input`, and `output`. The `soapbind:binding`, `soapbind:operation`, and `soapbind:body` elements, on the other hand, are protocol-specific. They are members of the namespace for the SOAP-WSDL binding, "http://schemas.xmlsoap.org/wsdl/soap/". Here's the same snippet from Listing 5-2 again, this time with the SOAP-specific elements in bold:

```
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  ...
  <!-- binding tells us which protocols and encoding styles are used -->
  <binding name="BookPrice_Binding" type="mh:BookQuote">
    <soapbind:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getBookPrice">
      <soapbind:operation style="rpc"
       soapAction=
       "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
        <input>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </input>
        <output>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </output>
    </operation>
  </binding>
  ...
</definitions>
```

~~The `soapbind:binding` and `soapbind:body` elements are responsible for expressing the SOAP-specific details of the Web service. For example, `soapbind:binding` tells us that the messaging style is RPC and that the network application protocol is HTTP. The `soapbind:body` element tells us that both the input and output messages use literal encoding. The SOAP-specific binding elements are discussed in more detail later in this chapter. The children of the `binding` element (`operation`, `input`, and `output`) map directly to the corresponding children of the `portType` element. Figure 5-4 shows the relationship between the `"BookQuote" portType` and the `"BookQuote_Binding" binding` element.~~

Figure 5-4. Binding to `portType` Mapping

~~Although the binding example given thus far uses SOAP-binding elements, the WSDL specification actually defines two other protocol-specific bindings, for HTTP and MIME. Listing 5-9 shows an example of an HTTP/URL encoding and a MIME payload binding.~~

~~Listing 5-9 Using the HTTP and MIME bindings~~

```
<!--HTTP/MIME-binding of BookQuote -->
<binding name="BookPrice_HttpMimeBinding" type="mh:BookQuote">
   <http:binding verb="GET"/>
   <operation name="getBookPrice">
      <http:operation location="alt.http.service"/>
      <input>
         <http:urlEncoded/>
      </input>
      <output>
         <mime:content type="text/html"/>
      </output>
   </operation>
</binding>
```

Although WSDL 1.1 allows the use of MIME and HTTP bindings as shown in Listing 5-9, the Basic Profile does not because they're poorly documented. Web services are required to use SOAP 1.1 binding, as illustrated in Listing 5-2.[BP]

~~This restriction will only be enforced by the Basic Profile 1.0. The WS-I has announced that it will explicitly extend support for SwA, including the WSDL MIME bindings in the next version of the Basic Profile, version 1.1. SwA is covered in Appendix E. SwA is still a subject of debate, however. Some leading SOAP authorities believe that SwA is the wrong solution to a rather simple problem. SwA relies on the MIME standard, which introduces a second packaging standard on top of the SOAP envelope. SwA critics argue that binary data can simply be encoded, using W3C's Hexadecimal or Base-64 built-in types. It seems likely however, that SwA will remain the de facto standard for attaching binary data to SOAP messages.~~

5.5.1 SOAP Binding

Several SOAP 1.1-specific binding elements are used in combination with the WSDL binding elements. These include `soapbind:binding`, `soapbind:operation`, `soapbind:body`, `soapbind:fault`, `soapbind:header`, and `soapbind:headerfault`. The `soapbind:binding` and `soapbind:body` elements are required, but the other elements are optional. ~~Using the `soapbind` prefix with the SOAP-binding namespace is a convention used in this book, but it's not required; you can use any prefix, just as with any XML namespace. The namespace assigned to that prefix must, however, be associated with the namespace defined by WSDL 1.1 for SOAP 1.1 bindings,~~ ~~`"http://schemas.xmlsoap.org/wsdl/soap/"`.~~

5.5.1.1 The `soapbind:binding` Element

~~The `soapbind:binding` element identifies the Internet protocol used to transport SOAP messages and the default messaging style (RPC or document) of its operations. The following snippet from Listing 5-2 shows the proper declaration of a `soapbind:binding` element.~~

```
<!-- binding tells us which protocols and encoding styles are used -->
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc"
     soapAction=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
      <input>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </input>
      <output>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </output>
  </operation>
</binding>
```

this operation name must be in corresponding portType. if not deployment error

declaring "style" attribute is optional. i tested.

Because J2EE Web Services supports only SOAP bindings, the `soapbind:binding` element must be declared in the WSDL `binding` element.[BP] In addition, the `style` attribute must be declared as either `"rpc"` or `"document"`; no other values are acceptable.[BP]

Because HTTP is the only transport protocol allowed by J2EE Web Services, the `transport` attribute must be declared to be HTTP, which means its value must be `"http://schemas.xmlsoap.org/soap/http/"`.[BP] This is the transport that corresponds to the SOAP-HTTP binding defined by SOAP 1.1. The `transport` attribute *must* be declared with an explicit value; there is no default.

if not there, deployment error. (i tested)

The requirement that you use the HTTP protocol does not prohibit use of HTTPS—HTTP 1.1 over SSL (Secure Sockets Layer). In WSDL, HTTPS is actually a part of the HTTP namespace. Whether vanilla HTTP or HTTPS is used depends on the schema declared by the `location` attribute of the `port` element, which is discussed later in Section 5.6.[BP]

i tried with HTTPS in both "location" and "transport". i could not get output DO:

<soap:operation> element can be empty that is without any attribute in rpc styled binding (i tested)

~~*It's possible that the Basic Profile w*~~
~~*besides HTTP eventually, including SMTP and TCP/IP*~~.

5.5.1.2 The `soapbind:operation` Element

even it is optional for document style operation, it ***MUST*** be declared RPC styled operation. if not method cannot identified by server. (i tested)

The ~~`soapbind:operation` element is required. It sp~~ a ~~specific operation and the value of the `SOAPAction` highlights the proper declaration of `soapbind:operation`.~~

```
<!-- binding tells us which protocols and encoding styles are used  -->
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc"
     soapAction=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
      <input>
        <soapbind:body use="literal"
         namespace="http://www.Monson-
      </input>
      <output>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </output>
  </operation>
</binding>
```

it has default value for both attribute, that why is optional

"soapAction" might be used by webservice container. in JSE, there is way to access HTTP headers, even entire httpRequest by using <filter>. (i tested)

According to BS-I, in SINGLE <binding> should be in "rpc" styled or "document" styled operation. but not mixed of both. But in J2EE it is possible mixed of both operation (i tested)

In WSDL 1.1 the `style` attribute is optional. It can be used to override the default messaging style declared by the `soapbind:binding` element—a capability that has been a source of interoperability problems. Accordingly, the Basic Profile requires that `style` attributes declared by `soapbind:operation` elements have the same value as the `style` attribute of their `soapbind:binding` element.[BP]

The `soapAction` attribute dictates the value that must be placed in the `SOAPAction` header field of the HTTP request message. You aren't required to declare the WSDL `soapAction` attribute; it can be omitted. You can also declare the `soapAction` attribute's value to be empty (indicated by two quotes), which is the same as omitting it.[BP] If this attribute is omitted or empty, then the `SOAPAction` HTTP header field must be present and must contain an empty string.[BP] The value of the `SOAPAction` HTTP header field must match, exactly, the value of the corresponding `soapAction` attribute in the `soapbind:operation` element. Listing 5-10 shows an HTTP SOAP request message whose `SOAPAction` header field matches the `soapAction` attribute declared in Listing 5-2.

SOAPActrion is HTTP header but not SOAP header

Listing 5-10 An HTTP SOA

```
1
m
arset="utf-8"
```

aaamam, how to get access HTTP header in endpoint without HttpRequest object ??
..
ANS:
we can access using <filter>

soapAction" attribute is just used to add value for HTTP Header named ***SOAPAction***. (but not soap header)
BUT it does not map anything to endpoint implementation.
if we want anything to pass in HTTP header we can use this and get in servlet, like in SiteMinder appplication.
But in real life no need this attribute
Acording to BP-1, this attribute muse be in soap Message, even it is omitted

so, the RPC based soap message will contain a parent element, which is name of operation.
..
Document based ,message will have element name, which is comes from XMLSchema
..

```
SOAPAction="http://www.Monson-Haefel.com/jwsbook/BookQuote/

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote">
   <soap:Body>
      <mh:getBookPrice>
          <isbn>0321146182</isbn>
      </mh:getBookPrice>
   </soap:Body>
</soap:Envelope>
```

~~You can find an example of an HTTP SOAP message that declares the~~ `SOAPAction` ~~header field to be empty in Chapter 4, Listing 4-26.~~

The style of messaging has a direct impact on how the body of the SOAP message is constructed, so declaring the correct `style`, `"rpc"` or `"document"`, is important. When you use RPC-style messaging, the `Body` of the SOAP message will contain an element that represents the operation to be performed. This element gets its name from the `operation` defined in the `portType`. ~~The `operation` element will contain zero or more parameter elements, which are derived from the `input` message's parts—each parameter element maps directly to a message part. An output message works exactly the same way. Figure 5-5 illustrates.~~

Figure 5-5. Mapping the `portType` Operation and Message Parts to an RPC-Style SOAP Message

```
<message name="GetBookPriceRequest">
    <part name="isbn" type="xsd:string" />
  </message>
  <message name="GetBookPriceResponse">
    <part name="price" type="xsd:float" />
  </message>
```

```
1.0" encoding="UTF-8"?>

tp://schemas.xmlsoap.org/soap/envelope/"
://www.Monson-Haefel.com/jwsbook/BookQuote">

kPrice>
>0321146182</isbn>
ookPrice>

>
```

ok.... how they will identify one method among them, when receive one or more Element in SOAP body
....
ANS:
it might be based on <java-xml-type-mapping> element and its class name as argument in a java method.
OR..
identify message by its part and identify operating by just identifed message. so it is still myth. actually no need to know .. i am not going to write container for jax-rpc !

so, style of message is used just who will be immediate child of soap BODY either name of operation or schema component

```
      </input>
      <output>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </output>
    </operation>
  </binding>
```

When you use document-style messaging, the XML document fragment will be the direct child of the `Body` element of the SOAP message.[BP] The operation is not identified.

soap:body can be without "use" attribute, since it has an default value is 'literal'. i tested

5.5.1.3 The `soapbind:body` Element

You may have noticed that attributes of the `soapbind:body` element change depending on whether you use RPC- or document-style messaging. The `soapbind:body` element has four kinds of attributes: `use`, `namespace`, `part`, and `encodingStyle`.

(`encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`)

The `use` attribute is required to be `"literal"`—and that value is assumed if the `soapbind:body` element fails to declare the `use` attribute.[BP] The `encodingStyle` attribute is never used at all, because WS-I-conformant Web services are based on the W3C XML schema, which is implied by the `use="literal"` declaration. Other encoding styles, like SOAP 1.1 Encoding, are not used. The `part` attribute specifies which `part` elements in the message definition are being used. The `part` attribute is necessary only if you are using a subset of the `part` elements declared by a message.

still i am not clear about "part" attribute. DO:

i could not pass sub-set msg

In `"rpc"`-style messages, the `namespace` attribute must be specified with a valid URI.[BP] The URI can be the same as the `targetNamespace` of the WSDL document, as in the following snippet from Listing 5-2.

it must be valid uri, but no need to be same as targernamespace of WSDL. it can any arbitrary uri (i tested)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="BookQuoteWS"
 targetNamespace="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:mh="http://www.Monson-Haefel.com/jwsbook/BookQuote"
 xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  ...
  <!-- binding tells us which protocols and enco...
  <binding name="BookPrice_Binding" type="mh:Boo...
    <soapbind:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/...
    <operation name="getBookPrice">
      <soapbind:operation style="rpc"
       soapAction=
       "http://www.Monson-Haefel.com/jwsbook/Boo...
        <input>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </input>
        <output>
          <soapbind:body use="literal"
           namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
        </output>
    </operation>
  </binding>
  ...
</definitions>
```

if <soap:body> has namespace attribute, it must be rpc styled operation. if not, it dont say any error,but method not identified. (i tested)

...

even operation is "rpc" styled and if not having "namespace" method is not identified.

.....

so, namespace attribute is must with rpc styled operation for the correct method to be identified (i t tested)

In contrast, document-style messages must *not* specify the `namespace` attribute in the `soapbind:body` element. The namespace of the XML document fragment is derived from its XML schema.[BP] Listing 5-11 shows the binding element of a document-style WSDL definition—notice that the `soap:body` element does not specify a `namespace` attribute.

Listing 5-11 A Document-Style Binding Doesn't Declare the `namespace` Attribute

```
<!-- binding tells us which protocols and encoding styles are used -->
<binding name="SubmitPurchaseOrder_Binding" type="mh:SubmitPurchaseOrder">
  <soapbind:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="submit">
    <soapbind:operation style="document"/>
      <input>
        <soapbind:body use="literal" />
      </input>
      <output>
        <soapbind:body use="literal" />
      </output>
  </operation>
</binding>
```

ONLY According to BS-I, soap fault, soap header, soap headerfault element must use "literal" and and its message part should be with "element" attribute.
....
but in j2ee it is possible (i tested)

5.5.1.4 The `soapbind:fault` Element

~~In addition to the soapbind:body element, a binding operation may also declare fault elements. The fault message elements are on a par with the input and output message elements. Listing 5-12 shows an example of a binding with fault and soapbind:fault elements.~~

Listing 5-12 Using the `soapbind:fault` Element

```
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc"
     soapAction=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
    <input>
      <soapbind:body use="literal"
       namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </input>
    <output>
      <soapbind:body use="literal"
       namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
    </output>
   <fault name="InvalidArgumentFault">
     <soapbind:fault name="InvalidArgumentFault" use="literal" />
    </fault>
  </operation>
</binding>
```

both "name" must be same. the same name should be in <portType> also, so, three place will have same name .(i tested)

The WSDL `fault` and `soapbind:fault` elements include a mandatory name attribute,[BP] which refers to a specific fault message declared in the associated `port Type`. ~~In Listing 5-12, the fault and soapbind:fault elements refer to the InvalidArgumentFault, which was defined in the BookQuote port and soapbind:fault element, as shown in the following snippet from Listing 5-7.~~

fault msg can have either 'type' or 'element' in its part. not prblm

```
<portType name="BookQuote">
  <operation name="getBookPrice">
     <input name="isbn" message="mh:GetBookPriceRequest"/>
     <output name="price" message="mh:GetBookPriceResponse"/>
     <fault name="InvalidArgumentFault" message="mh:InvalidArgumentFault"/>
  </operation>
</portType>
```

Every name of <fault> element of a operation must be matched with binding <fault>

An operation may have zero or more `fault` elements, each with its own `soapbind:fault` element. Each `soapbind:fault` element may declare a `use` attribute. If it does, the value must be `"literal"`. If it doesn't, the value is `"literal"` by default.[BP]

ONLY According to BS-I, soap fault, soap header, soap headerfault element must use "literal" and and its message part should be with element.
....
but in j2ee it is possible

5.5.1.5 The `soapbind:header` Element

In Chapter 4 you learned that a SOAP message may have header blocks. WSDL explicitly identifies a SOAP header block by using the `soapbind:header` element in the binding's `input` element, its `output` element, or both. As an example we can create a binding that describes the `message-id` header block as in Listing 5-13.

soap:header can zero or more time (i tested)

soap:header and soap:headerFault can be associated with <input> or <output> but not with <fault>

usually, we can add or process SOAP header **without** touch WSDL file, using SAAJ api and <handler> element.
....
<soap:header> is alternative way to say in WSDL itself

Listing 5-13 Defining a SOAP Header Block

```
<types>
  <xsd:schema targetNamespace=
   "http://www.Monson-Haefel.com/jwsbook/BookQuote"
   xmlns="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="message-id" type="string" />
  </xsd:schema>
</types>

<!-- message elements describe the input and output parameters -->
<message name="Headers">
  <part name="message-id" element="mh:message-id" />
</message>
<message name="GetBookPriceRequest">
  <part name="isbn" type="xsd:string" />
</message>
<message name="GetBookPriceResponse">
  <part name="price" type="xsd:float" />
</message>

<!-- portType element describes the abstract interface of a Web service -->
<portType name="BookQuote">
  <operation name="getBookPrice">
    <input name="isbn" message="mh:GetBookPriceRequest"/>
    <output name="price" message="mh:GetBookPriceResponse"/>
  </operation>
</portType>

<!-- binding tells us which protocols and encoding styles are used -->
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc"
```

soap:header msg can have either type or element. and also, it can have complex typed element or type. (i tested)

```
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc"
     soapAction=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
      <input>
        <soapbind:header message="mh:Headers" part="message-id"
         use="literal" />
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </input>
      <output>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </output>
  </operation>
</binding>
```

"part" attribute is mandatory

printing mistake

The `message` and `part` attributes used by the `soapbind:header` element refer to the specific message `part` used for the fault. The `part` referred to must use an `element` attribute.[BP] In other words, you can't base a header block on a type definition, it must be based on a top-level element defined in the `types` element, or imported in some other XML schema document or WSDL document. The `use` attribute is always equal to `"literal"`, whether it's explicitly declared or not.[BP]

ONLY According to BS-I, soap fault, soap header, soap headerfault element must use "literal" and and its message part should be with element.
....
but in j2ee it is possible

5.5.1.6 The `soapbind:headerfault` Element

The `soapbind:headerfault` element describes a header block-specific fault message. As you learned in Chapter 4, if there is a response message, any header block-specific faults must be returned in its `Header` element. WSDL maintains this scoping by requiring that `soapbind:headerfault` elements be nested in their associated headers. For example, a fault message specific to the `message-id` header block would be declared as in Listing 5-14, which adds a `soapbind:headerfault` to the WSDL document defined in Listing 5-13.

even if there is no <output>, still header fault can be used. Not any error. (i tested)
.....
But, <headerfault> comes to useful only if that operation has <output> message only.

Listing ... **headerfault Element**

```
<!-- ... input and output parameters -->
 <mes...
  <pa... "mh:detailMessage" />
</message>
...
<!-- binding tells us which protocols and encoding styles are u...
<binding name="BookPrice_Binding" type="mh:BookQuote">
  <soapbind:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getBookPrice">
    <soapbind:operation style="rpc"
     soapAction=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote/GetBookPrice"/>
      <input>
        <soapbind:header message="mh:Header" use="literal">
          <soapbind:headerfault message="mh:Headers" use="literal" />
```

so, the soap:headerfault can be nested ONLY with soap:header of <input> binding and only the operation has <output> message.

"part" must be in header fault also. book missed it. (i tested)

```
        </soapbind:header>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </input>
      <output>
        <soapbind:body use="literal"
         namespace="http://www.Monson-Haefel.com/jwsbook/BookQuote" />
      </output>
  </operation>
</binding>
```

The `soapbind:headerfault` element has the same requirements as the `soapbind:header`. It must declare a `message` attribute that points to the appropriate `message` definition, and its `use` attribute must be equal to `"literal"`, whether explicitly declared or by default.[BP]

5.6 WSDL Implementation: The `service` and `port` Elements

The `service` element cont port element cannot be without any address sub element rt elements, each of which represents a different Web service. The `port` element a specific `binding`. The following snippet shows a complete service definition from List 5-13.

more than one <service> element possible in single WSDL file (i tested)

```
<service name="BookPriceService">
  <port name="BookPrice_Port" binding="mh:BookPrice_Binding">
    <soapbind:address location=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote" />
  </port>
</service>
```

A `service` may have more than one `port` element, each of which assigns a URL to a specific binding. It's even possible for two or more `port` elements to assign different URLs to the same `binding`, which might be useful for load balancing or failover. Listing 5-15 shows a `service` element that contains three `port` elements, two of which refer to the same `binding`.

Listing 5-15 Defining a `service` with Multiple `port` Elements

```
<service name="BookPriceService">
  <port name="BookPrice_Port" binding="mh:BookPrice_Binding">
    <soapbind:address location=
     "http://www.Monson-Haefel.com/jwsbook/BookQuote" />
  </port>
  <port name="BookPrice_Failover_Port" binding="mh:BookPrice_Binding">
    <soapbind:address location=
     "http://www.monson-haefel.org/jwsbook/BookPrice" />
  </port>
  <port name="SubmitPurchaseOrder_Port"
   binding="mh:SubmitPurchaseOrder_Binding">
    <soapbind:address location=
     "https://www.monson-haefel.org/jwsbook/po" />
  </port>
</service>
```

same

5.6.1 The `soapbind:address` Element

The `soapbind:address` element is pretty straightforward; it simply assigns an Internet address to a SOAP binding via its `location` attribute (its only attribute). Although WSDL allows any type of address (HTTP, FTP, SMTP, and so on), the Basic Profile allows only those URLs that use the HTTP or HTTPS schema.[BP] For example, in Listing 5-15 the first two `port` e not, just <service> . ohhh ddress for the `location` attribute, while the third `port` declares an HTTPS address.

Two or more `port` elements within the same WSDL document must not specify exactly the same URL value for the `location` attribute of the `soapbind:address`.[BP]

the same URL address cannot be shared by more than one <port> in the SAME WSDL file not just <service> element

5.7 WS-I Conformance Claims

A WS-I **conformance claim** can be assigned to any WSDL definition, asserting adherence to the WS-I Basic Profile 1.0 specification. Child elements inherit their parents' conformance claims; for example, a `portType`'s claim that it conforms to the BP also applies to all the `operation` and `message` definitions associated with that `portType`. The best place to put a conformance claim is inside the `port` definition, because it applies to all the other definitions associated with that `port` (`binding`, `portType`, `operation`, and `message`).

get example of this usage from net in next round DO:

5.8 Wrapping Up

~~The most important thing to remember about WSDL is that it provides a precise, structured, and standard format for describing Web services. This is advantageous for both vendors creating code generators and developers using SOAP APIs. The precision and strict structure of WSDL allows vendors to offer tools that automatically generate callable interfaces to a specific Web service, and enables developers using SOAP APIs to construct, deliver, and process SOAP messages correctly when using lower-level APIs like SAAJ~~.

~~In many cases you will not deal directly with WSDL documents, because code generators such as JAX-RPC providers will create convenient language-specific call interfaces for invoking Web services. In addition, existing interfaces can be used with tools to generate WSDL documents, so in many cases you may not be exposed to the contents of WSDL documents at all. While a detailed knowledge of WSDL document structure isn't necessary when using code generators, it is important for you to understand the organization and purpose of WSDL documents if you wish to truly master Web services. You have to understand WSDL to construct and exchange SOAP messages properly when using SOAP APIs, and these tools are often important when code generators are not available or are not robust enough to support your messaging requirements.~~

~~WSDL 1.1 is a lot more flexible than the WS-I Basic Profile allows, but the requirements of the Basic Profile make WSDL documents more portable and the SOAP messages they describe interoperable. Although the WS-I does a great job of constraining WSDL and therefore increasing interoperability~~, it doesn't state where the WSDL documents may be stored. I~~n some cases the location of the WSDL document might be relative to the access URL of the Web service itself, in other cases it may not. The Basic Profile does, however, tell us how to refer to WSDL documents from a UDDI registry. Use of UDDI is optional, but if it is used, it must be implemented according to a strict set of guidelines. UDDI is the subject of the next part of this book.~~

finished on 15 / May / 2009 with video