2. WSDL 2.0 Basics

2.1 Getting Started: The GreatH Hotel Example

This section introduces the basic concepts used in WSDL 2.0 through the description of a hypothetical hotel reservation service. We start with a simple scenario, and later add more requirements to illustrate how more advanced WSDL 2.0 features may be used.

2.1.1 Example Scenario: The GreatH Hotel Reservation Service

Hotel GreatH (a fictional hotel) is located in a remote island. It has been relying on fax and phone to provide room reservations. Even though the facilities and prices at GreatH are better than what its competitor offers, GreatH notices that its competitor is getting more customers than GreatH. After research, GreatH realizes that this is because the competitor offers a Web service that permits travel agent reservation systems to reserve rooms directly over the Internet. GreatH then hires us to build a reservation Web service with the following functionality:

- **CheckAvailability.** To check availability, the client must specify a check-in date, a check-out date, and room type. The Web service will return a room rate (a floating point number in USD) if such a room is available, or a zero room rate if not. If any input data is invalid, the service should return an error. Thus, the service will accept a checkAvailability message and return a checkAvailabilityResponse or invalidDataFault message.

- **MakeReservation.** To make a reservation, a client must provide a name, address, and credit card information, and the service will return a confirmation number if the reservation is successful. The service will return an error message if the credit card number or any other data field is invalid. Thus, the service will accept a makeReservation message and return a makeReservationResponse or invalidCreditCardFault message.

We know that we will later need to build a complete system that supports transactions and secured transmission, but initially we will implement only minimal functionality. In fact, to simplify our first example, we will implement only the CheckAvailability operation.

The next several sections proceed step-by-step through the process of developing a WSDL 2.0 document that describes the desired Web service. However, for those who can't wait to see a complete example, here is the WSDL 2.0 document that we'll be creating.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<description
```
<documentation>
This document describes the GreatH Web service. Additional application-level requirements for use of this service -- beyond what WSDL 2.0 is able to describe -- are available at http://greath.example.com/2004/reservation-documentation.html
</documentation>

<types>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://greath.example.com/2004/schemas/resSvc"
xmlns="http://greath.example.com/2004/schemas/resSvc">
<xs:element name="checkAvailability" type="tCheckAvailability"/>
<xs:complexType name="tCheckAvailability">
<xs:sequence>
<xs:element name="checkInDate" type="xs:date"/>
<xs:element name="checkOutDate" type="xs:date"/>
<xs:element name="roomType" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:element name="checkAvailabilityResponse" type="xs:double"/>
<xs:element name="invalidDataError" type="xs:string"/>
</xs:schema>
</types>

@interface name = "reservationInterface">
<fault name = "invalidDataFault"
        element = "ghns:invalidDataError"/>
<operation name="opCheckAvailability"
        pattern="http://www.w3.org/2003/05/soap-envelope"
        style="http://www.w3.org/2003/05/soap-envelope"
        wsdlx:safe = "true">
<input messageLabel="In"
        element="ghns:checkAvailability" />
<output messageLabel="Out"
2.1.2 Defining a WSDL 2.0 Target Namespace

Before writing our WSDL 2.0 document, we need to decide on a WSDL 2.0 target namespace URI for it. The WSDL 2.0 target namespace is analogous to an XML Schema target namespace. Interface, binding and service names that we define in our WSDL 2.0 document will be associated with the WSDL 2.0 target namespace, and thus will be distinguishable from similar names in a different WSDL 2.0 target namespace. (This will become important if using WSDL 2.0's import or interface inheritance mechanisms.)

The value of the WSDL 2.0 target namespace must be an absolute URI. Furthermore, it should be dereferenceable to a WSDL 2.0 document that describes the Web service that the WSDL 2.0 target namespace is used to describe. For example, the GreatH owners should make the WSDL 2.0 document available from this URI. (And if a WSDL 2.0 description is split into multiple documents, then the WSDL 2.0 target namespace should resolve to a master document that includes all the WSDL 2.0 documents needed for that service description.) However, there is no absolute requirement for this URI to be dereferenceable, so a WSDL 2.0 processor must not depend on it being dereferenceable.
This recommendation may sound circular, but bear in mind that the client might have obtained the WSDL 2.0 document from anywhere -- not necessarily an authoritative source. But by dereferencing the WSDL 2.0 target namespace URI, a user should be able to obtain an authoritative version. Since GreatH will be the owner of the service, the WSDL 2.0 target namespace URI should refer to a location on the GreatH Web site or otherwise within its control.

Once we have decided on a WSDL 2.0 target namespace URI, we can begin our WSDL 2.0 document as the following empty shell.

```
Example 2-2. An Initial Empty WSDL 2.0 Document

<?xml version="1.0" encoding="utf-8" ?>
<description
     xmlns="http://www.w3.org/ns/wsdl"
     targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
   . . .>
   . . .
</description>
```

2.1.2.1 Explanation of Example

```
<description
   xmlns="http://www.w3.org/ns/wsdl"
   targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
   . . .>
   . . .
</description>
```

Every WSDL 2.0 document has a `description` element as its top-most element. This merely acts as a container for the rest of the WSDL 2.0 document, and is used to declare namespaces that will be used throughout the document.

```xml
xmlns="http://www.w3.org/ns/wsdl"
```

This is the XML namespace for WSDL 2.0 itself. We assign it as the default namespace for this example by not defining a prefix for it. In other words, any unprefixed elements in this example are expected to be WSDL 2.0 elements (such as the `description` element).

```xml
targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
```

This defines the WSDL 2.0 target namespace that we have chosen for the GreatH reservation service, as described above. Note that this is not an actual XML namespace declaration. Rather, it is a WSDL 2.0 attribute whose purpose is analogous to an XML Schema target namespace.

```xml
```

This is an actual XML namespace declaration for use in our GreatH service description. Note that this is the same URI that was specified above as the value of the `targetNamespace` attribute. This will allow us later to use the `tns:` prefix in QNames, to refer to the WSDL 2.0 target namespace of the GreatH service. (For more on QNames see [XML Namespaces] section 3 Qualified Names.)
Now we can start describing the GreatH service.

2.1.3 Defining Message Types

We know that the GreatH service will be sending and receiving messages, so a good starting point in describing the service is to define the message types that the service will use. We'll use XML Schema to do so, because WSDL 2.0 processors are likely to support XML Schema at a minimum. However, WSDL 2.0 does not prohibit the use of some other schema definition language.

WSDL 2.0 allows message types to be defined directly within the WSDL 2.0 document, inside the types element, which is a child of the description element. (Later we'll see how we can provide the type definitions in a separate document, using XML Schema's import mechanism.) The following schema defines checkAvailability, checkAvailabilityResponse and invalidDataError message types that we'll need.

In WSDL 2.0, all normal and fault message types must be defined as single elements at the topmost level (though of course each element may have any amount of substructure inside it). Thus, a message type must not directly consist of a sequence of elements or other complex type.

Example 2-3. GreatH Message Types

```xml
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    ...>
    ...

    <types>
        <xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://greath.example.com/2004/schemas/resSvc"
            xmlns=http://greath.example.com/2004/schemas/resSvc">
            <xs:element name="checkAvailability" type="tCheckAvailability"/>
            <xs:complexType name="tCheckAvailability">
                <xs:sequence>
                    <xs:element name="checkInDate" type="xs:date"/>
                    <xs:element name="checkOutDate" type="xs:date"/>
                    <xs:element name="roomType" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </types>
</description>
```
2.1.3.1 Explanation of Example


We've added another namespace declaration. The ghns namespace prefix will allow us (later, when defining an
interface) to reference the XML Schema target namespace that we define for our message types. Thus, the URI we
specify must be the same as the URI that we define as the target namespace of our XML Schema types (below) --
_not_ the target namespace of the WSDL 2.0 document itself.

targetNamespace="http://greath.example.com/2004/schemas/resSvc"

This is the XML Schema target namespace that we've created for use by the GreatH reservation service. The
checkAvailability, checkAvailabilityResponse and invalidDataError element names will be associated with this
XML Schema target namespace.

checkAvailability, checkAvailabilityResponse and invalidDataError

These are the message types that we'll use. Note that these are defined to be XML elements, as explained above.

Although we have defined several types, we have not yet indicated which ones are to be used as message types for a
Web service. We'll do that in the next section.

2.1.4 Defining an Interface

WSDL 2.0 enables one to separate the description of a Web service's abstract functionality from the concrete details of
how and where that functionality is offered. This separation facilitates different levels of reusability and distribution of work
in the lifecycle of a Web service and the WSDL 2.0 document that describes it.

A WSDL 2.0 _interface_ defines the abstract interface of a Web service as a set of abstract _operations_, each operation
representing a simple interaction between the client and the service. Each operation specifies the types of messages that
the service can send or receive as part of that operation. Each operation also specifies a message exchange _pattern_ that
indicates the sequence in which the associated messages are to be transmitted between the parties. For example, the
_in-out_ pattern (see WSDL 2.0 Predefined Extensions [WSDL 2.0 Adjuncts] section 2.2.3 In-Out) indicates that if the client
sends a message *in* to the service, the service will either send a reply message back *out* to the client (in the normal case) or it will send a fault message back to the client (in the case of an error). We will explain more about message exchange patterns in 2.4.4.3 Understanding Message Exchange Patterns (MEPs)

For the GreatH service, we will (initially) define an interface containing a single operation, `opCheckAvailability`, using the `checkAvailability` and `checkAvailabilityResponse` message types that we defined in the `types` section. We’ll use the **in-out** pattern for this operation, because this is the most natural way to represent a simple request-response interaction. We could have instead (for example) defined two separate operations using the **in-only** and **out-only** patterns (see WSDL 2.0 Predefined Extensions [WSDL 2.0 Adjuncts] section 2.2.1 In-Only and section 2.2.5 Out-Only), but that would just complicate matters for the client, because we would then have to separately indicate to the client developer that the two operations should be used together as a request-response pair.

In addition to the normal input and output messages, we also need to specify the fault message that we wish to use in the event of an error. WSDL 2.0 permits fault messages to be declared within the `interface` element in order to facilitate reuse of faults across operations. If a fault occurs, it terminates whatever message sequence was indicated by the message exchange pattern of the operation.

Let’s add these to our WSDL 2.0 document.

```xml
Example 2-4. GreatH Interface Definition

<?xml version="1.0" encoding="utf-8" ?>
<description>
  xmlns="http://www.w3.org/ns/wsdl"
  targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
  xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">
  ...
  <interface  name = "reservationInterface" >
    ...
    <fault name = "invalidDataFault"
      element = "ghns:invalidDataError" />
    <operation name="opCheckAvailability"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri"
      wsdlx:safe = "true">
      <input messageLabel="In"
        element="ghns:checkAvailability" />
```
2.1.4.1 Explanation of Example

Interfaces are declared directly inside the `description` element. In this example, we are declaring only one interface, but in general a WSDL 2.0 document may declare more than one interface. Thus, each interface must be given a name that is unique within the set of interfaces defined in this WSDL 2.0 target namespace. Interface names are tokens that must not contain a space or colon (".").

Faults are declared inside the `interface` element. In this example, we are declaring only one fault, but in general a WSDL 2.0 document may declare more than one fault. Thus, each fault must be given a name that is unique within the set of faults defined in this WSDL 2.0 target namespace. Fault names are tokens that must not contain a space or colon (".").

Operations are declared inside the `interface` element. In this example, we are declaring only one operation, but in general a WSDL 2.0 document may declare more than one operation. Thus, each operation must be given a name that is unique within the set of operations defined in this WSDL 2.0 target namespace. Operation names are tokens that must not contain a space or colon (".").
wsdlx:safe="true" >

This line indicates that this operation will not obligate the client in any way, i.e., the client can safely invoke this operation without fear that it may be incurring an obligation (such as agreeing to buy something). This is further explained in 2.4.4 Interface Operations.

<input messageLabel="In"

The input element specifies an input message. Even though we have already specified which message exchange pattern the operation will use, a message exchange pattern represents a template for a message sequence, and in theory could consist of multiple input and/or output messages. Thus we must also indicate which potential input message in the pattern this particular input message represents. This is the purpose of the messageLabel attribute. Since the in-out pattern that we've chosen to use only has one input message, it is trivial in this case: we simply fill in the message label "In" that was defined in WSDL 2.0 Predefined Extensions [WSDL 2.0 Adjuncts] section 2.2.3 In-Out for the in-out pattern. However, if a new pattern is defined that involve multiple input messages, then the different input messages in the pattern could then be distinguished by using different labels.

element="ghns:checkAvailability" />

This specifies the message type for this input message, as defined previously in the types section.

<output messageLabel="Out"

This is similar to defining an input message.

<outfault ref="tns:invalidDataFault" messageLabel="Out"/>

This associates an output fault with this operation. Faults are declared a little differently than normal messages. The ref attribute refers to the name of a previously defined fault in this interface -- not a message schema type directly. Since message exchange patterns could in general involve a sequence of several messages, a fault could potentially occur at various points within the message sequence. Because one may wish to associate a different fault with each permitted point in the sequence, the messageLabel is used to indicate the desired point for this particular fault. It does so indirectly by specifying the message that will either trigger this fault or that this fault will replace, depending on the pattern. (Some patterns use a message-triggers-fault rule; others use a fault-replaces-message rule. See WSDL 2.0 Predefined Extensions [WSDL 2.0 Adjuncts] section 2.1.2 Message Triggers Fault and section 2.1.1 Fault Replaces Message.)

Now that we've defined the abstract interface for the GreatH service, we're ready to define a binding for it.

2.1.5 Defining a Binding

Although we have specified what abstract messages can be exchanged with the GreatH Web service, we have not yet specified how those messages can be exchanged. This is the purpose of a binding. A binding specifies concrete message format and transmission protocol details for an interface, and must supply such details for every operation and fault in the
In the general case, binding details for each operation and fault are specified using operation and fault elements inside a binding element, as shown in the example below. However, in some cases it is possible to use defaulting rules to supply the information. The WSDL 2.0 SOAP binding extension, for example, defines some defaulting rules for operations. (See Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts [WSDL 2.0 Adjuncts], Default Binding Rules.)

In order to accommodate new kinds of message formats and transmission protocols, bindings are defined using extensions to the WSDL 2.0 language, via WSDL 2.0's open content model. (See 4.1 Extensibility for more on extensibility.) WSDL 2.0 Part 2 [WSDL 2.0 Adjuncts] defines binding extensions for SOAP 1.2 [SOAP 1.2 Part 1: Messaging Framework] and HTTP 1.1 [IETF RFC 2616] as predefined extensions, so that SOAP 1.2 or HTTP 1.1 bindings can be easily defined in WSDL 2.0 documents. However, other specifications could define new binding extensions that could also be used to define bindings. (As with any extension, other WSDL 2.0 processors would have to know about the new constructs in order to make use of them.)

For the GreatH service, we will use SOAP 1.2 as our concrete message format and HTTP as our underlying transmission protocol, as shown below.

```
Example 2-5. GreatH Binding Definition

<?xml version="1.0" encoding="utf-8" ?>
<description
 xmlns="http://www.w3.org/ns/wsdl"
 targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
 xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
 xmlns:soap= "http://www.w3.org/2003/05/soap-envelope">
 . . .
 <types>
 . . .
 </types>
 <interface name = "reservationInterface" >
 . . .
 </interface>
 <binding name="reservationSOAPBinding"
 interface="tns:reservationInterface"
 type="http://www.w3.org/2003/05/soap"
 wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/" >
 <operation ref="tns:opCheckAvailability"
 wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
```

2.1.5.1 Explanation of Example

We've added two more namespace declarations. This one is the namespace for the SOAP 1.2 binding extension that is defined in WSDL 2.0 Part 3 [SOAP 1.2 Part 1: Messaging Framework]. Elements and attributes prefixed with wsoap: are constructs defined there.

This namespace is defined by the SOAP 1.2 specification itself. The SOAP 1.2 specification defines certain terms within this namespace to unambiguously identify particular concepts. Thus, we will use the soap: prefix when we need to refer to one of those terms.

Bindings are declared directly inside the description element. The name attribute defines a name for this binding. Each name must be unique among all bindings in this WSDL 2.0 target namespace, and will be used later when we define a service endpoint that references this binding. WSDL 2.0 uses separate symbol spaces for interfaces, bindings and services, so interface "foo", binding "foo" and service "foo" are all distinct.

This is the name of the interface whose message format and transmission protocols we are specifying. As discussed in 2.5 More on Bindings, a reusable binding can be defined by omitting the interface attribute. Note also the use of the tns: prefix, which refers to the previously defined WSDL 2.0 target namespace for this WSDL 2.0 document. In this case it may seem silly to have to specify the tns: prefix, but in 3.1 Importing WSDL we will see how WSDL 2.0's import mechanism can be used to combine components that are defined in different WSDL 2.0 target namespaces.

This specifies what kind of concrete message format to use, in this case SOAP 1.2.

This attribute is specific to WSDL 2.0's SOAP binding extension (thus it uses the wsoap: prefix). It specifies the
underlying transmission protocol that should be used, in this case HTTP.

```xml
<operation ref="tns:opCheckAvailability"

This is not defining a new operation; rather, it is referencing the previously defined opCheckAvailability operation in order to specify binding details for it. This element can be omitted if defaulting rules are instead used to supply the necessary information. (See the SOAP binding extension in WSDL 2.0 Part 2 [WSDL 2.0 Adjuncts] section 4.3 Default Binding Rules.)

```xml
wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response">

This attribute is also specific to WSDL 2.0's SOAP binding extension. It specifies the SOAP message exchange pattern (MEP) that will be used to implement the abstract WSDL 2.0 message exchange pattern (in-out) that was specified when the opCheckAvailability operation was defined.

When HTTP is used as the underlying transport protocol (as in this example) the wsoap:mep attribute also controls whether GET or POST will be used as the underlying HTTP method. In this case, the use of wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response" causes GET to be used by default. See also 2.5.7 HTTP GET Versus POST: Which to Use?.

```xml
<fault ref="tns:invalidDataFault"

As with a binding operation, this is not declaring a new fault; rather, it is referencing a fault (invalidDataFault) that was previously defined in the opCheckAvailability interface, in order to specify binding details for it.

```xml
wsoap:code="soap:Sender"/>

This attribute is also specific to WSDL 2.0's SOAP binding extension. This specifies the SOAP 1.2 fault code that will cause this fault message to be sent. If desired, a list of subcodes can also be specified using the optional wsoap:subcodes attribute.

2.1.6 Defining a Service

Now that our binding has specified how messages will be transmitted, we are ready to specify where the service can be accessed, by use of the service element.

A WSDL 2.0 service specifies a single interface that the service will support, and a list of endpoint locations where that service can be accessed. Each endpoint must also reference a previously defined binding to indicate what protocols and transmission formats are to be used at that endpoint. A service is only permitted to have one interface. (See 5.4 Multiple Interfaces for the Same Service for further discussion of this limitation.)

Here is a definition for our GreatH service.

```xml
Example 2-6. GreatH Service Definition

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

```
2.1.6.1 Explanation of Example

<service name="reservationService">

This defines a name for this service, which must be unique among service names in the WSDL 2.0 target namespace. The name attribute is required. It allows URIs to be created that identify components in WSDL 2.0 description. (See WSDL 2.0 Core Language [WSDL 2.0 Core] appendix C IRI References for WSDL 2.0 constructs.)

interface="tns:reservationInterface">

This specifies the name of the previously defined interface that these service endpoints will support.
<endpoint name="reservationEndpoint"
  <description>  
  <documentation>
    This document describes the GreatH Web service. Additional application-level requirements for use of this service -- beyond what WSDL 2.0 is able to describe -- are available at http://greath.example.com/2004/reservation-documentation.html
    </documentation>
    . . . 
  </description>
</endpoint>
<documentation>

This element is optional, but a good idea to include. It can contain arbitrary mixed content.


The most important thing to include is a pointer to any additional documentation that a client developer would need in order to use the service.

This completes our presentation of the GreatH example. In the following sections, we will move on to look into more details of various aspects of WSDL 2.0 specification.

2.2 WSDL 2.0 Infoset, Schema and Component Model

In computer science theory, a language consists of a (possibly infinite) set of sentences, and each sentence is a finite string of literal symbols or characters. A language specification must therefore define the set of sentences in that language, and, to be useful, it should also indicate the meaning of each sentence. Indeed, this is the purpose of the WSDL 2.0 specification.

However, instead of defining WSDL 2.0 in terms of literal symbols or characters, to avoid dependency on any particular character encoding, WSDL 2.0 is defined in terms of the XML Infoset [XML Information Set]. Specifically, a WSDL 2.0 document consists of a description element information item (in the XML Infoset) that conforms to the WSDL 2.0 specification. In other words, a sentence in the WSDL 2.0 language is a description element information item that obeys the additional constraints spelled out in the WSDL 2.0 specification.

Since an XML Infoset can be created from more than one physical document, a WSDL 2.0 document does not necessarily correspond to a single physical document: the word "document" is used figuratively, for convenience. Furthermore, since WSDL 2.0 provides import and include mechanisms, a WSDL 2.0 document may reference other WSDL 2.0 documents to facilitate convenient organization or reuse. In such cases, the meaning of the including or importing document as a whole will depend (in part) on the meaning of the included or imported document.

The XML Infoset uses terms like "element information item" and "attribute information item". Unfortunately, those terms are rather lengthy to repeat often. Thus, for convenience, this primer often uses the terms "element" and "attribute" instead, as a shorthand. It should be understood, however, that since WSDL 2.0 is based on the XML Infoset, we really mean "element information item" and "attribute information item", respectively.

2.2.1 WSDL 2.0 Infoset

The following diagram gives an overview of the XML Infoset for a WSDL 2.0 document.
2.2.2 WSDL 2.0 Schema

The WSDL 2.0 specification supplies a normative WSDL 2.0 schema, defined in [XML Schema Structures], which can be used as an aid in validating WSDL 2.0 documents. We say "as an aid" here because WSDL 2.0 specification [WSDL 2.0 Core] often provides further constraints to the WSDL 2.0 schema. In addition to being valid with the normative schema, a WSDL 2.0 document must also follow all the constraints defined by the WSDL 2.0 specification.
2.2.2.1 WSDL 2.0 Element Ordering

This section gives an example of how WSDL 2.0 specification constrains the WSDL 2.0 schema about the ordering of top WSDL 2.0 elements.

Although the WSDL 2.0 schema does not indicate the required ordering of elements, the WSDL 2.0 specification (WSDL 2.0 Part 1 [WSDL 2.0 Core] section "XML Representation of Description Component") clearly states a set of constraints about how the child elements of the description element should be ordered. Thus, the order of the WSDL 2.0 elements matters, even though the WSDL 2.0 schema does not capture this constraint.

The following is a pseudo-content model of description.

```
<description>
  <documentation />??
  [ <import /> | <include /> ]*
  <types />??
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

In other words, the children elements of the description element should be ordered as follows:

- An optional documentation comes first, if present.
- then comes zero or more elements from among the following, in any order:
  - include
  - import
  - extensions
- An optional types follows
- Zero or more elements from among the following, in any order:
  - interface
  - binding
  - service
  - extensions.

Note the term "extension" is used above as a convenient way to refer to namespace-qualified extension elements. The namespace name of such extension elements must not be "http://www.w3.org/ns/wsdl".

2.2.3 WSDL 2.0 Component Model
The WSDL 2.0 Infoset model above illustrates the required structure of a WSDL 2.0 document, using the XML Infoset. However, the WSDL 2.0 language also imposes many semantic constraints over and above structural conformance to this XML Infoset. In order to precisely describe these constraints, and as an aid in precisely defining the meaning of each WSDL 2.0 document, the WSDL 2.0 specification defines a component model as an additional layer of abstraction above the XML Infoset. Constraints and meaning are defined in terms of this component model, and the definition of each component includes a mapping that specifies how values in the component model are derived from corresponding items in the XML Infoset. The following diagram gives an overview of the WSDL 2.0 components and their containment hierarchy.

Figure 2-2. WSDL 2.0 Components Containment hierarchy
In general, the WSDL 2.0 component model parallels the structure of the required XML Infoset illustrated above. For example, the Description, Interface, Binding, Service and Endpoint components correspond to the description, interface, binding, service, and endpoint element information items, respectively. Since WSDL 2.0 relies heavily on the component model to convey the meaning of the constructs in the WSDL 2.0 language, you can think of the Description component as representing the meaning of the description element information item, and hence, it represents the meaning of the WSDL 2.0 document as a whole.

Furthermore, each of these components has properties whose values are (usually) derived from the element and attribute information item children of those element information items. For example, the Service component corresponds to the service element information item, so the Service component has an {endpoints} property whose value is a set of Endpoint components corresponding to the endpoint element information item children of that service element information item. (Whew!)

### 2.2.3.1 WSDL 2.0 Import and Include

The WSDL 2.0 component model is particularly helpful in defining the meaning of import and include elements. The include element allows you to assemble the contents of a given WSDL 2.0 namespace from several WSDL 2.0 documents that define components for that namespace. The components defined by a given WSDL 2.0 document consist of those whose definitions are contained in the document and those that are defined by any WSDL 2.0 documents that are included in it via the include element. The effect of the include element is cumulative so that if document A includes document B and document B includes document C, then the components defined by document A consist of those whose definitions are contained in documents A, B, and C.

In contrast, the import element does not define any components. Instead, the import element declares that the components whose definitions are contained in a WSDL 2.0 document for a given WSDL 2.0 namespace refer to components that belong to a different WSDL 2.0 namespace. If a WSDL 2.0 document contains definitions of components that refer to other namespaces, then those namespaces must be declared via an import element. The import element also has an optional location attribute that is a hint to the processor where the definitions of the imported namespace can be found. However, the processor may find the definitions by other means, for example, by using a catalog.

After processing any include elements and locating the components that belong to any imported namespaces, the WSDL 2.0 component model for a WSDL 2.0 document will contain a set of components that belong to the document's WSDL 2.0 namespace and any imported namespaces. These components will refer to each other, usually via QName references. A WSDL 2.0 document is invalid if any component reference cannot be resolved, whether or not the referenced component belongs to the same or a different namespace.

We will cover a lot more about how to use WSDL 2.0 import and include in [3.1 Importing WSDL](#).

### 2.3 More on Message Types