2003-05-13 Action Item

Ümit Yalçınalp
Oracle Corporation

Referencing Web Services:

This document illustrates different approaches to defining a web service reference within a WSDL document by rewriting wsdl:endpoint proposal given by Arthur Ryman [1][2]. We start with some observations before exploring the solution to referencing web services.

A web service endpoint URI is not enough to designate a web service reference. This is due to the fact that one endpoint address can be used by multiple endpoints (hence bindings) to represent different bindings and the related interface. Therefore, the only way to uniquely identify a web service is by using the following components:

(URI, binding, interface, targetResource)

- It is implicit that the service element refers to a single target resource and hence the endpoints that are contained in the service elements are equivalent with respect to accessing the resource.

- When there are more than one service elements that share the same targetResource as explicitly designated by the targetResource attribute, they are equivalent if the interfaces in those service elements that share the resource are the same.

It means that:

(URI1, B1, I, TR) equiv (URI2, B2, I, TR)

Note that a client who has access to a *service element's contents* following the layered structure of WSDL and XML schema documents in a bottom up manner, will be able to access a web service at runtime. From a clients perspective, the three components are necessary to define a reference to a web service. (Please note that the interface can be inferred from the binding, but listed here for completeness).

Requirements:

We observe that the following two requirements are related:

1. Statically defining a service reference in a WSDL document when referred definitions of Web services are available (the interface and the binding of the reference is known).
2. Defining the structure of a service reference that the client may receive. The solution to this requirement should establish how a reference can be used dynamically. We discuss further requirements in this context later as it is important to define the structure of a reference that can be used dynamically.

In this document, different alternatives to solving the reference problem for (1) modifying Arthur's proposal is presented. One of the motivations for this exercise to get rid of the XPath expressions in the definition of the `wsdl:endpoint` elements.

As a result of this analysis, we also present an alternative way of representing a web service reference which allows more dynamic treatment of web service references and can be used at runtime.

These two approaches are orthogonal and the solution to (1) also is applicable in (2) as necessary.

Static Definition of WS-Reference in WSDL by using wsdl:endpoint or schema annotations:

Arthur Ryman presents a strongly typed, statically defined solution to representing a web service reference[2]. In his document, he designates the three components that constitute a reference and gives a REST example for generating references to a web service.

The idea in Arthur's proposal that a URI is essentially a designator for a reference along with the interface and the binding. *This is essentially what a service element would designate in a WSDL document.*

- Schema designates the URI for the endpoint.
- The interface of the web service that exposes a reference designates the interface of the referred service. An XPath expression designates where in the instance document the reference would occur in relationship to a message part.
- The binding of the web service that exposes a reference designates the binding of the referred service.

The complete example can be found in [1] and [2]. For the sake of completeness, we are going to represent the schema of the Part and Parts that is used by Arthur's example *replacing the xlink:href attributes for the sake of simplicity.*

The following schema holds to define Part and Parts elements:

```
<xs:element name=”Parts”>
    <xs:complexType>
        <xs:sequence>
         <xs:element ref=”tns:Part” minOccurs=”0”
                     maxOccurs=”unbounded”/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name=”Part”>
    <xs:complexType>
        <xs:attribute name=”id” type=”xs:integer” use=”required”/>
        <xs:attribute name=”reference” type=”xs:anyURI” use=”required”/>
    </xs:complexType>
</xs:element>
```

WSDL Segments for obtaing a list of references to Part elements from [2]:

(1):

```
<wsdl:message name="getPartListInput"/>
<wsdl:message name="getPartListOutput">
   <wsdl:part name="return" element="p:Parts"/>
</wsdl:message>
<wsdl:interface name="partListInterface">
   <wsdl:operation name="GET">
      <wsdl:input message="tns:getPartListInput">
      <wsdl:output message="tns:getPartListOutput">
      <wsdl:endpoint name="partURI" part="return"
            xpath="/p:Parts/Part/@reference"
            interface="tns:partInterface"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:interface>
```

(2):

```
<wsdl:binding name="PartListHTTPBinding" type="tns:partListInterface">
   <http:binding verb="GET"/>
   <wsdl:operation name="GET">
      <http:operation location=""/>
      <wsdl:input>
         <http:urlEncoded/>
      </wsdl:input>
      <wsdl:output>
      <wsdl:endpoint name="partURI" binding="tns:partHTTPBinding"/>
         <mime:mimeXml part="return"/>
      </wsdl:output>
   </wsdl:operation>
 </wsdl:binding>
```

(3):

```
<wsdl:service name="PartListService">
   <wsdl:port name="PartListHTTPPort" binding="tns:PartListHTTPBinding">
      <http:address location="http://www.parts-depot.com/parts"/>
   </wsdl:port>
</wsdl:service>
```

Although not recorded in the minutes [3], one of the motivating factors for this was to get rid of designating XPath expressions in wsdl:endpoint elements both in the interface and in the binding (1) and (2).

We explore how we could accomplish this in three alternate ways below:

(I)

Use a new type to designate a reference URI. This was explored a bit in the f2f, but not fully.

We introduce a new type **ws-reference-uri** in wsdl namespace**.**

```
<xs:simpleType name="ws-reference-uri">
     <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
```

Using this new type directly in the schema will provide that the reference URI is an endpoint address:

```
<xs:element name="Part">
    <xs:complexType>
        <xs:attribute name="id" …/>
        <xs:attribute name="reference"
                      type="wsdl:ws-reference-uri"
                      use="required"/>
    </xs:complexType>
<xs:element>
```

However, this is not enough as a solution. This is due to the fact that a message part can contain multiple references, therefore the need for using a designator such as XPath or SCD to point to the intended interface/binding does not go away in the wsdl:endpoint declaration within the interface component.

For example if the part **return** should contain a complex structure that contains two references, one to a Part and another to a Machine, the interface information would need to be defined for each reference separately for the same message part. So this is not enough to get rid of path designators from both interface and from the binding. In rethinking XPath, we observe that it designates specific locations with respect to the instance document although the intent is to designate a location in the Schema (type) definition. For this purpose, SCD appears to be a better option as the domain for SCD is a Schema, not the instance document itself.

In order to get rid of this problem, we would declare a *separate type for each interface reference* that extends the ws-reference-uri as follows.

```
<xs:simpleType name="PartReference">
     <xs:restriction base="wsdl:ws-reference-uri"/>
</xs:simpleType>

<xs:element name="Part">
    <xs:complexType>
        <xs:attribute name="id" …/>
        <xs:attribute name="reference"
                      type="tns:PartReference"
                      use="required"/>
    </xs:complexType>
</xs:element>
```

This approach provides a typed marker in the Schema . However, each different reference "type" must be designated by a new type derived from **wsdl:ws-reference-uri** in order to make this work. For example a PartReference and MachineReference would be two different typed references.

The wsdl:endpoint definition would need to define for each different reference in the schema the interface it corresponds to in a specific part component. For example in (1') below, the occurance of **type="tns:PartReference"** in the message part **return** indicates a reference to an endpoint associated with the interface **tns:partInterface**. *Note that this approach still requires us to express the relationship in the wsdl:endpoint element, but gets rid of using Xpath/SCD.*

(1'):

```
<wsdl:message name="getPartListInput"/>
<wsdl:message name="getPartListOutput">
   <wsdl:part name="return" element="p:Parts"/>
</wsdl:message>
<wsdl:interface name="partListInterface">
   <wsdl:operation name="GET">
      <wsdl:input message="tns:getPartListInput">
      <wsdl:output message="tns:getPartListOutput">
      <wsdl:endpoint name="partURI" part="return"
            interface ="tns:partInterface" type="tns:PartReference"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:interface>
```

Since there may be multiple occurances of a reference that corresponds to a different interface type, the wsdl:endpoint declaration has to be made *for each different interface type and the corresponding schema type marker.* Basically this approach indicates a typed reference by deriving from a special data type, `ws-reference-uri`, defined in the wsdl namespace. There is a one to one mapping between the derived marker type and the intended interface type. Otherwise, the WSDL document is in error as it would be illegal to use a marker type for two different interfaces.

(II)

Another way of accomplishing this goal is to use a schema annotation in the derived type to designate the target interface and/or binding.

We use two new elements, `interface-ref` and `binding-ref` of type xs:QName to designate an interface and a binding reference defined in wsdl namespace as a schema annotation:

```
(*)
<xs:simpleType name="PartReference">
     <xs:annotation>
        <xs:appInfo>
          <wsdl:interface-ref>tns:partInterface</wsdl:interface-ref>
          <wsdl:binding-ref>tns:partHTTPBinding</wsdl:binding-ref> (**)
       </xs:appInfo>
     </xs:annotation>
     <xs:restriction base="wsdl:ws-reference-uri"/>
</xs:simpleType>
```

An alternate, but equivalent way of doing this is to use the `wsdl:interface-ref` elements as an annotation for the elements that are declared by using the type `wsdl:ws-reference-uri` type.

(**) Most likely, the binding information may not be desired to be supplied at design of the interface(s). This is when the interfaces may be defined before an appropriate binding is chosen. For this use case, the `wsdl:binding-ref` element should not occur in the schema annotation, i.e. `wsdl:binding-ref` elements in the annotations are not necessary.

In this case, the `wsdl:endpoint` element in the binding definition will remain and will designate a specific binding in reference to a reference type as illustrated below. The type attribute is used to designate where the reference occurs in the schema. (Note that the name of the `wsdl:endpoint` element is not very useful in this case as the `wsdl:endpoint` element only occurs in the binding. Therefore, it should be regarded as optional.)

(2'):

```
<wsdl:binding name="PartListHTTPBinding" type="tns:partListInterface">
   <http:binding verb="GET"/>
   <wsdl:operation name="GET">
      <http:operation location=""/>
      <wsdl:input>
         <http:urlEncoded/>
      </wsdl:input>
      <wsdl:output>
      <wsdl:endpoint name="partURI" binding="tns:partHTTPBinding"
                      type="tns:PartReference"/>
         …
      </wsdl:output>
   </wsdl:operation>
 </wsdl:binding>
```

This approach gets rid of the `wsdl:endpoint` elements from the interface and potentially the binding components in a WSDL document. However, it has one major drawback: We have created a circular dependency from the schema component to the a WSDL component. This has the drawback of carrying backward pointer of the type as to how it should be used.

(III)

Another approach is to mark a type in the schema as a reference to something else by using a new attribute, `interface-ref (a QName)` defined in wsdl namespace. This approach is similar to the approach taken in [5] for incorporating xmime:MediaType attributes to a schema.

Consider the following declarations in WSDL:

```
<xs:complexType name="ws-reference-uri" >
    <xs:simpleContent>
      <xs:extension base="xs:anyURI"/>
    </xs:simpleContent>
 </xs:complexType>

<xs:attribute name="interface-ref" >
    <xs:simpleType>
      <xs:restriction base="xs:QName"/>
    </xs:simpleType>
</xs:attribute>
```

with following declaration for the Part:

```
<xs:element name=”Part”>
    <xs:complexType>
        <xs:sequence>
            <xs:element name=”reference”
                type=”wsdl:ws-reference-uri”
                wsdl:interface-ref=”tns:PartInterface”/>
       </xs:sequence>
        <xs:attribute name=”id” type=”xs:integer”
                      use=”required”/>
    </xs:complexType>
<xs:element>
```

In this approach, the `wsdl:interface-ref` attribute marks the declaration of the reference with the intended interface for the reference.

Comparison of these approaches:

Approach (II) has a circular dependency. Approach (III) also has the same problem and also requires special handling of the WSDL namespace in the schema itself. In both instances, Schema depends on WSDL constructs. (III) would require extensions to the existing Schema processors and tools that depend on them that is specific to processing WSDL documents.

(I) is cleaner as it requires the declaration of a reference type for each type of interface that needs to be referenced.

There is one drawback. We got rid of XPath expressions (or SCD) only partially. This is because this model has the following limitation. Our Part/Parts example is a bad one to illustrate this. If two or more bindings as occurances of the same schema type (hence interface) in the same design need to occur as part of a message, each binding still must be declared statically. Assume that a message needs to contain two different references to a bank account where each reference uses a different binding for the bank account, we need to use either a separate type derivation for the bank account interface for each binding or use the XPath expression approach.

A Different Approach to Representing Web Service References:

In the previous section, we discussed alternative ways to solve one problem: Statically defining the interface and the binding of a web service in relationship to an endpoint URI. In this case, the endpoint URI is considered to be the minimum to communicate to a web service at runtime. In this section, we tackle the orthogonal problem as a result of this analysis: An alternate way of representing a reference.

We make the observation that we made earlier in this document in the first section: *The definition of a service component is all we need to designate a web service reference.*

We think that the definition of a web service reference should satisfy the following requirements:

- It should contain enough information in itself to indicate the endpoint it is referring to, not a URI alone.
- It should also allow the receiver to be able to use alternate, but equivalent endpoints, depending on the receivers capabilities without a fixed endpoint address.
- It should not depend on a specific binding, but should allow indicating a specific binding at design time, in WSDL.
- It is applicable statically and can also be used dynamically.

Instead of a typed URI, we think the approach is more general and can accommodate dynamic references by satisfying these requirements.

Based on our observation alone, we think that a web service reference is simply the service component itself, along with an optional endpoint name that designates the preferred endpoint to use the reference. Simple, the following complex type that designates web service reference:

```
<xs:complexType name="ws-reference">
       <xs:sequence>
           <xs:element name="service" type="xs:QName"/>
           <xs:element name="endpoint" type="xs:NCName" minOccurs="0"/>
           <xs:element name="wsdl-location" type="xs:anyURI"
                       minOccurs="0"/>
           <xs:any processContents="lax"
                   minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
       <xs:anyAttribute namespace="##any" processContents="lax" />
</xs:complexType>
```

From the service element, we can always infer the interface and the endpoints at runtime. From the endpoint, a reference always exposes its URI, hence the address of the endpoint. The wsdl-location element indicates the location of the WSDL document.

This structure is the minimal we need to define a service reference. However, it can be used both statically and dynamically as it contains all the information including the actual reference to the service. Hence there are two cases to consider:

(A). When the reference's interface is known at design time:

The solutions presented in the previous section of this document provides different ways to statically relate a web service reference and the interface of a service. The same techniques also apply even if a different mechanism for formulating a web service reference is used.

- A new type that is derived from ws-reference is created, i.e.

  ```
  <complexType name="PartReference">
     <extension ref="wsdl:ws-reference>
  </complexType>
  ```

- The techniques illustrated in I, II, or III must be used to embed the interface intended for this reference. This is necessary, because the reference type itself, just like a URI, does not contain information about the interface for the reference. This information needs to be provided within the WSDL document. As a result, the WSDL provided in (1') would be used to designate the reference in the interface by using the `wsdl:endpoint` element, or the annotation as declared in (*) or the example in (III) (replacing the ws-reference with ws-reference-uri) would still apply.

- The binding refers to the specific type that designates the reference. In this case, however, the binding information is *optional*. Again using `wsdl:endpoint` element, the binding information may be present, if desired. With this design, however, it is optional because the service element contains multiple binding(s) that can be used. Further, this design does not suffer the same limitation as mentioned in the comparison section as different bindings may be present to the same interface within references of the same type.

(B) When the reference's interface and/or binding is not known at design time.

In this case, the same type that we have defined can be used directly *without using* `wsdl:endpoint` elements. The WSDL will directly use the reference as is without indicating what the reference is for because late binding for the type is possible using the service element.

In comparison to the typed URI approach, a receiver of a reference has enough information to interpret the endpoint address and its type, binding, etc since service element definition is used. Using this approach, alternative endpoints can be used by the receiver of the service. However, typed URIs by themselves do not have this information and can not be used as a runtime structure, like the `wsdl:ws-reference` type. Further, a binding for the reference is optional in the WSDL document with this approach. This question has been currently posed to the WSD wg in [4].

The definition of the `wsdl:ws-reference` type assumes `xs:QName` resolution, hence the availability of the service element and the WSDL document for the receiver of a reference. We provided an optional element, `wsdl-location,` for this purpose. Although it is not covered in this document, there may be alternate ways to obtain the WSDL document, such as using attachments.

We at Oracle looked at this option and concluded that it is more general and it allows dynamicly defining bindings and interfaces. Other information such as runtime properties that may be necessary for a runtime reference can also be defined by using this type, similar to an IOR that represents a runtime reference structure. One other useful information may be to add an optional operation name that can be used by the client. The addional information is subject to further discussion and we wanted to start the discussion in that direction. In order to accommodate extensibility, we provided extensibility options in the type definition.

Our current approach describes it as a reference using the WSDL concepts directly. Further, the `ws-reference` type can be used at both in the WSDL and can be used as the basis for on-the-wire representation of the reference.

Summary:

In this document, we discussed several different ways of interjecting interface and binding information that is necessary for declaring a reference in the WSDL document.

We also propose an alternate way of declaring a service reference that we would like to be considered as the basis for defining the definition of a web service reference. It allows both static and dynamic definition of a reference.

We think that using a typed URI by itself is not enough to fully define a reference, however the techniques that apply in designating a static interface in WSDL is applicable to both Arthur's proposal as well as to our proposal.

References:

[1] http://lists.w3.org/Archives/Public/www-ws-desc/2003May/0043.html

[2] http://lists.w3.org/Archives/Public/www-ws-desc/2003Apr/0088.html

[3] Discussion in May f2f

http://lists.w3.org/Archives/Public/www-ws-desc/2003May/0051.html
[4] http://lists.w3.org/Archives/Public/www-ws-desc/2003May/0004.html

[5] Proposed Infoset Addendum to SOAP Messages with Attachments
http://www.gotdotnet.com/team/jeffsch/paswa/paswa61.html