UDDI Spec TC

Using WSDL in a UDDI Registry, Version 1.08

Document identifier:

uddi-spec-tc-bp-using-wsdl-v108-20021110

Location:

http://www.oasis-open.org/committees/uddi-spec/doc/bp/uddi-spec-tc-bp-using-wsdl-v108-20021110.htm

Editors:

John Colgrave, IBM
Karsten Januszewski, Microsoft

Contributors:

Francisco Curbera, IBM
David Ehnebuske, IBM
Dan Rogers, Microsoft

Abstract:

This document describes the current Best Practice for constructing UDDI entities from, or relating to, WSDL descriptions of web services.

Status:

This document is updated periodically on no particular schedule.

Committee members should send comments on this best practice to the uddi-spec@lists.oasis-open.org list. Others should subscribe to and send comments to the uddi-spec-comment@lists.oasis-open.org list. To subscribe, send an email message to uddi-spec-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this best practice, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the UDDI Spec TC web page (http://www.oasis-open.org/committees/uddi-spec/).

Table of Contents

1 Introduction

The Universal Description Discovery and Integration (UDDI) specification provides a platform-independent way of describing services, discovering businesses, and integrating business services using the Internet. The UDDI data structures provide a framework for the description of basic business and service information, and architects an extensible mechanism to provide detailed service access information using any standard description language. Many such languages exist in specific industry domains and at different levels of the protocol stack. The Web Services Description Language (WSDL) is a general purpose XML language for describing the interface, protocol bindings and the deployment details of network services. WSDL complements the UDDI standard by providing a uniform way of describing the abstract interface and protocol bindings of arbitrary network services. The purpose of this document is to clarify the relationship between the two, describe how WSDL can be used to help create UDDI business service descriptions.

1.1 Special Note on Version 1.08

In this version, the format of the fragment identifier used in the overviewURL to refer to a specific wsdl:binding in a WSDL document (See *tModels for WSDL Service Interface Definitions*) has been changed. See *Appendix A* for information on the implication of this change.

2 Relevant UDDI Structures

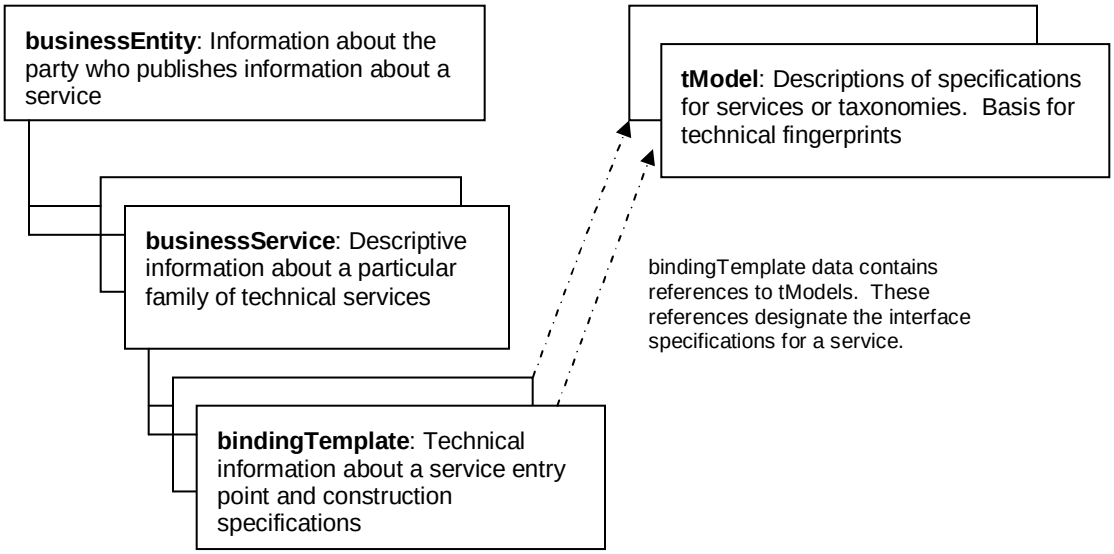
As an aid to understanding the sections ahead, we provide here a brief overview of two UDDI data structures that are particularly relevant to the use of WSDL in the context of a UDDI registry: the tModel, also known as the service type definition, and the businessService. Refer to [1] for additional information.

2.1 tModels

tModels provide the ability to describe compliance with a specification, a concept, or a shared design. tModels have various uses in the UDDI registry. We are interested here in the use of tModels to represent technical specifications like wire protocols, interchange formats and sequencing rules. When a particular specification is registered with the UDDI repository as a tModel, it is assigned a unique key, which is then used in the description of service instances to indicate compliance with the specification.

2.2 The businessService

Services are represented in UDDI by the **businessService** data structure, and the details of how and where the service is accessed are provided by one or more nested **bindingTemplate** structures.



A bindingTemplate specifies a network endpoint address (in the **accessPoint** element) and a stack of tModels describing the service.

```
<businessService>
   (...)
   <bindingTemplates>

      <bindingTemplate>
         (...)
         <accessPoint urlType="http"> http://www.etc.com/</accessPoint>
         <tModelnstanceDetails>
            <tModelnstanceInfo tModelKey="...">

            </tModelnstanceInfo>
            (...)
         </tModelnstanceDetails>
      </bindingTemplate>
      (...)

   </bindingTemplates>
</businessService>
```

3 WSDL Service Descriptions

WSDL service descriptions can be structured in multiple ways. However, if the reusable information is separated from the information that is specific to a given service instance, the use of WSDL and UDDI together becomes particularly simple. This section explains how WSDL enables this form of document organization. Additional details can be found in [2].

3.1 Structure of WSDL Service Descriptions

The **import** element in WSDL allows the separation of elements of the description of a service into two parts, referred to here as “service interface definition” and “service implementation definition” WSDL service information. Typically, information common to a certain category of business services, such as message formats, portTypes (abstract interfaces), and protocol bindings, are included in the reusable portion, while information pertaining to a particular service endpoint (i.e., port definition) is included in the service implementation definition portion. Within the context of UDDI we will be concerned only with the reusable portion of the WSDL service information.

4 Authoring UDDI Service Descriptions

We now consider how WSDL can support the creation of UDDI businessService entries. We summarize the process in three major steps.

1. The first step is to create the WSDL service interface definition. Typically, industry groups will define a set of service types, and describe them with one or more service interface definition WSDL documents. The service interface definition will include service interfaces and protocol bindings, and will be made publicly available. The WSDL service interface definitions are then registered as UDDI tModels; the overviewDoc field in each new tModel will point to the corresponding WSDL document (see Section: “tModels for WSDL service interface definitions” for details). We refer to such tModels as “wsdlSpec tModels”.

2. Next, programmers will build services that conform to the industry standard service definitions. Either manually or using appropriate UDDI-aware tooling, programmers will retrieve the tModel description of the industry standard definition, and (following the overviewDoc link) obtain the corresponding WSDL definition document. WSDL-aware tooling, in turn, can help generate an implementation that supports the standard interfaces and bindings.

3. Finally, the new service must be deployed and registered in the UDDI repository. Either manually or using WSDL and UDDI-aware tooling, a UDDI businessService data structure is created, and then registered. Typically when using WSDL and UDDI-aware tools, service deployment information (some type of “deployment descriptor” document) will be generated at that same time.

The information contained in the new businessService references the implemented standards and provides additional deployment details:

- A bindingTemplate is created for each service access endpoint. The network address of the access point is encoded in the accessPoint element.

- One tModelInstanceInfo is created in the bindingTemplate for each tModel that is relevant to the service end point being described, in particular, for every wsdlSpec tModel that defines interfaces and bindings supported by the service.

4.1 Registering and Referencing WSDL Definitions in UDDI

The development and registration of services described in the previous section assumes that some WSDL service information is registered or embedded in the UDDI registry. In this section we provide a more detailed explanation of how this information is embedded and referenced in UDDI. We focus on two aspects of this problem:

1. Registration of WSDL service interface definitions as UDDI tModels.
2. Reference to reusable wsdlSpec tModels in bindingTemplates.

4.2 tModels for WSDL Service Interface Definitions

WSDL service interface definitions are intended to describe many service instances, and it is consequently natural to register them as tModels. In the case when the description comprises more than one WSDL document, one tModel should be created for each. Each such tModel must be classified, using the uddi-org:types taxonomy, as being of type “wsdlSpec” and must have an overviewDoc whose overviewURL points to the relevant WSDL document. An example is outlined below.

```
<tModel authorizedName="..." operator="..." tModelKey="...">
   <name>StockQuote Service</name>
   <description xml:lang="en">
      WSDL description of a standard stock quote service interface
   </description>
   <overviewDoc>
      <description xml:lang="en">WSDL source document.
       </description>
       <overviewURL>
          http://stockquote-definitions/stq.wsdl
       </overviewURL>
   </overviewDoc>
   <categoryBag>
  <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
                  keyName="uddi-org:types"
                  keyValue="wsdlSpec"/>
   </categoryBag>
</tModel>
```

When a tModel refers to a WSDL document as in this example, it refers to the entire content of the document, including all of its bindings. It is often convenient to have a tModel refer to a single binding. This may be accomplished by having the overviewURL contain as a fragment identifier [3] an XPointer [4] to the binding.[1] For example, the overviewURL above could have been coded as:

```
<overviewURL>
 http://stockquote-definitions/stq.wsdl#xmlns(wsdl=http://schemas.xmlsoap.org/wsdl/)
xpointer(//wsdl:binding[@name='StockQuoteSoapBinding'])
</overviewURL>
```

to make the tModel refer specifically to the StockQuoteSoapBinding.

[1] This is a change to Version 1.07 of this document. See *Appendix A* for an explanation of the change and of its consequences.

4.3 UDDI Binding Templates and wsdlSpec tModels

tModelInstanceInfo structures in the bindingTemplate use the tModelKey attribute to refer to the technical specifications that are required to interact with that service endpoint. When WSDL and UDDI are used together, the tModel referred to should be one of type wsdlSpec, that is, one whose overviewDoc is a WSDL service interface definition, as explained in the previous section. One tModelInstanceInfo structure must be created for each wsdlSpec tModel containing standard definitions that are relevant to the service being defined.

5 Example

We consider here a modified version of the example presented in Section 1.2 of the WSDL specification [2] to illustrate the mapping presented in the previous section. For simplicity, we will assume that the WSDL service interface definition consists of only one document, which contains type, message, portType, and binding definitions. The WSDL service interface definition is given below:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2001/XMLSchema ">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
   </types>

   <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding"
             type="tns:StockQuotePortType">
        <soap:binding style="document"
           transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation
```

```
            soapAction="http://example.com/GetLastTradePrice"/>
         <input>
             <soap:body use="literal"
                namespace="http://example.com/stockquote.xsd
                encodingStyle=
                   "http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output>
             <soap:body use="literal"
                namespace="http://example.com/stockquote.xsd"
                encodingStyle=
                   "http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
       </operation>
   </binding>

</definitions>
```

Assuming that we place the WSDL service interface definition at http://example.com/stockquote.wsdl, the corresponding tModel would be published as follows:

```
<tModel authorizedName="..." operator="..." tModelKey="...">
   <name>StockQuote Service</name>
   <description xml:lang="en">
      WSDL description of a standard stock quote service interface
   </description>
   <overviewDoc>
      <description xml:lang="en">
         WSDL source document.
  </description>
  <overviewURL>
         http://example.com/stockquote.wsdl
  </overviewURL>
    </overviewDoc>
    <categoryBag>
  <keyedReference tModelKey=" uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
     keyName="uddi-org:types"
     keyValue="wsdlSpec"/>
    </categoryBag>
</tModel>
```

The bindingTemplate structure of the service would then be published such that its tModelInstanceInfo references the tModel of the WSDL service interface definition above:

```
<businessService businessKey="..." serviceKey="...">
   <name>StockQuoteService</name>
   <description> (...) </description>
   <bindingTemplates>
      <bindingTemplate>
         (...)
         <accessPoint urlType="http">
            http://example.com/stockquote
         </accessPoint>
         <tModelnstanceDetails>
            <tModelnstanceInfo tModelKey="...">

            </tModelnstanceInfo>
         <tModelnstanceDetails>
      </bindingTemplate>
   </bindingTemplates>
</businessService>
```

6 References

1. “UDDI Version 2.03 Data Structure Reference”, July 19, 2002. Available at http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.pdf.
2. Web Services Description Language (WSDL) 1.1, March 15, 2000. Available at http://www.w3.org/TR/wsdl
3. “Uniform Resource Identifiers (URI): Generic Syntax”, IETF Draft Standard, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt
4. “XPointer xpointer() Scheme”, W3C Working Draft 10 July 2002. Available at http://www.w3.org/TR/2002/WD-xptr-xpointer-20020710/

Appendix A. Fragment Identifier Format

In this version, the format of the fragment identifier used in the overviewURL to refer to a specific wsdl:binding in a WSDL document (See *tModels for WSDL Service Interface Definitions*) has been changed to conform to the XPointer xpointer() scheme. This change is not compatible with the fragment identifiers specified in Version 1.07 of this document. Publishers are urged to change to the new form of fragment identifier.

In Version 1.07, the fragment identifier was defined to be the value of the name attribute of the wsdl:binding being referred to. References of that form, when interpreted as an XPointer typically do not refer to anything and definitely do not refer to wsdl:binding elements.

To help inquirers who need to resolve WSDL document overviewURLs with fragment identifiers, the following procedure should be used.

1. Interpret the fragment identifier as an XPointer. If the result is a reference to a wsdl:binding, that is the binding the overviewURL refers to.
2. Otherwise interpret the fragment as the value of the name attribute of a wsdl:binding (i.e., interpret it according to Version 1.07 of this document). If the result is a wsdl:binding, that is the binding the overviewURL refers to.
3. Otherwise, the overviewURL has been incorrectly coded and the reference is invalid.

Appendix B. Change History

1. June 25, 2001: Changed reference 1 to point to UDDI V2.o Data Structure Reference. Changed reference 2 to point to the WSDL 1.1 TR.
2. 21 May 2002: Corrected example taken from [2]
3. 31 October 2002: Changed fragment identifier to XPointer.
4. 10 November 2002: Changed reference 1 to point to UDDI V2.03 Data Structure Reference.

Appendix C. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.