Usage of document/literal wrapped pattern in WSDL design

Vivek Gupta
April 21, 2011

A WSDL (Web Services Description Language) binding style can be either RPC or document, but the wrapped-document/literal pattern is the one, which blends the best out of both these styles.

Introduction

While defining a highly interoperable Web Service in a WSDL using document/literal style of messaging, a “pattern” which is known as wrapped-document/literal, is followed across service providers. This pattern takes the advantage of best features of both the RPC/Literal as well as Document/Literal messaging in WSDL by:

1. Inserting the SEI (Service Endpoint Interface) Operation name in SOAP Message – Feature of RPC/literal.

A WSDL describes a web service by defining its interface, the operations on that interface, their binding to the transport protocol, and the network address they are available at. A WSDL binding describes how a web service is bound to a messaging protocol, that is, what is the format or structure of the Messages being exchanged with a Service, and the transport protocol over which this Message will be sent. The most popular messaging protocol in web services today, is the SOAP messaging protocol (The other option is to use the REST-based Messages). For a web service using SOAP messaging protocol, the binding “style” can be either RPC (remote-procedure-call) or document style. The binding style being chosen is important from the point of view of the WSDL definition of Input & Output messages for a Web Service Operation & the corresponding SOAP Request/Response message structure. Only "literal" (each element or type is schema defined and serialization follows definition from W3C XML Schema) way of representing the message is supported in WS-I (Web Service – interoperability organization) recommendation for these two binding styles.

In document/literal style of messaging, there exists a pattern which is known as wrapped-document/literal. This is just a pattern, and is not a part of WSDL specification. This pattern has a mention in JSR 224 (JAX-WS: Java API for XML based web services).
This article provides the rules for this pattern in WSDL and intends to stress on the usage of this pattern in WSDL Design. This style/pattern is discussed in the following sections.

**Rules for document/literal – Wrapped pattern style of SOAP messaging**

The rules for the "wrapped" convention to be followed during WSDL Design:

1. **Only "One" Part Definition in the Input & Output Messages in WSDL**
   "Wrapped" is a form of document/literal. When defining a WS-I compliant document/literal service, there can be at most one body part in your input message and at most one body part in your output message. You do *not* define each method parameter as a separate part in the message definition. (The parameters are defined in the WSDL "types" section, instead).

2. **"Part" Definitions are wrapper elements**
   Each part definition must reference an element (not a type, type is used in RPC) defined to make it document style of messaging. This element definition can be imported, or included in the types section of the WSDL document. These element definitions are "wrapper" elements (hence the name of this convention). Define the input and output parameters as element structures within these wrapper elements.

3. **Child Elements of "Part" Element Type will be SEI Method parameter**
   An input wrapper element must be defined as a complex type that is a sequence of elements. Each child element-type in that sequence will be generated (while using code generation tool on WSDL) as a parameter of the operation in the service interface.

4. **Input Wrapper Element name should match with Operation name**
   The name of the input wrapper element must be the same as the web service operation name in WSDL.

5. **<Output Wrapper Element Name> = <Operation Name> + "Response"**
   The name of the output wrapper element could be (but doesn't have to be) the operation name appended with "Response" (e.g., if the operation name is "add", the output wrapper element should be called "addResponse").

6. **In the WSDL Binding section, soap:binding style = "document"**
   Since, the style is document/literal for this wrapped pattern, hence in the binding definition, the soap:binding should specify style="document" (although this is the default value, so the attribute may be omitted), and the soap:body definitions must specify use="literal" and nothing else. You must not specify the namespace or encodingStyle attributes in the soap:body definition.

The illustration of this pattern is done through the diagrams from which style of **WSDL should I use?**

In listing 1 below you can see the wrapper element "myMethod" matches the operation name "myMethod" in portType section.

**Listing 1. Document/literal wrapped WSDL for myMethod**

```xml
<types>
...
</types>
```
The SOAP Request Message as shown below in listing 2.

**Listing 2. Document/literal wrapped SOAP message for myMethod**

```
<SOAP-ENV:Body>
  <myMethod>
    <x>-0</x>
    <y>3.14</y>
  </myMethod>
</SOAP-ENV:Body>
```

The following is a snapshot of Java SEI generated from the WSDL using "wsimport" facility from JWSDP 2.0. The WSDL used document/literal-wrapped pattern. As can be noticed, the method name will match the input element name.

**Listing 3. Java SEI from wsimport run on document/literal wrapped WSDL for myMethod**

```java
/**
 * This class was generated by JAXWS SI.
 * JAX-WS RI 2.0-b62-ea3
 * Generated source version: 2.0
 *
 */

@WebService(name ="PT", wsdlLocation = "PT.wsdl")
public interface PT {

  /**
   * ...
   */

  @param y
```
Advantages of document/literal – wrapped pattern

The following advantages can be considered by following this pattern -

1. **Wide-Acceptance of Document/Literal-Wrapped style amongst web service providers**
   Wrapped Document/Literal pattern provides highly interoperable web service design and has been accepted in the service provider community as a de facto standard while defining the Web Services. The wrapped pattern has evolved from Microsoft® environments. Given the fact that
   1. Microsoft does not have explicit support for RPC-Literal and
   2. Microsoft tools by default generate the WSDL using this wrapped pattern,
      It inherently increases the popularity of this style/pattern when it comes to interoperability.
      Dual Advantage –
      
      1. The complete SOAP Message under soap:Body can be validated against schema and
      2. This pattern is "always" compliant with WS-I by allowing only one child element under soap:Body.

This high interoperability index of this pattern has resulted into a wider acceptance by the industry as compared to other styles of messaging in Web Service such as RPC/Literal or Document/Literal un-wrapped. Also, IBM WebSphere® tools generate the WSDL with this pattern while using Bottom-up approach in developing web services.

2. **Advantage for SOAP Engine to "Dispatch" SOAP Request to appropriate Web Service Provider**
   The SOAP Engine has an advantage with this pattern since it has a clear and simple naming convention available during un-marshaling (XML to Java/C# etc) same as it is available in RPC style. This advantage may save the SOAP Engine from extra execution of matching the parameters in the input SOAP Request message (non WS-Address)
Figure 1. SOAP Message Dispatching to Endpoint Implementation

With this pattern, the SOAP Engine will use wsdl:portType to determine the Interface and the Method name to whom the SOAP Message should be dispatched. The industry SOAP Engine pioneers (for example Apache Axis2) dispatching process has a mapping of the first child element under soap:Body in the SOAP Message with the operation name.

The deployment descriptor for the Web Service contains the information on wrapped pattern. If two different methods in the Web Service have similar input parameter names, then it becomes increasingly difficult for the SOAP Engine to determine the exact operation name to which the SOAP message should be dispatched to. For wrapped pattern, the operation name is mapped so, there is no additional restriction on the input parameter names to be different in different methods for a given web service.

3. Optimal Approach
   This style is generally regarded as an optimal approach that enhances maintainability, comprehension, portability, and, to a certain extent, performance. The performance is better in this pattern mainly due to the following reasons –
   1. No Encoding used (advantage of "literal" usage).
   2. Direct parsers & optimized validation of complex data structures of SOAP Messages (advantage of Document style of messaging).
   3. Certain tests have also claimed better performance over RPC/Literal.

4. Better choice for web service operations with multiple Input Parameters
   If you have a method which takes in more than a parameter, and you wish to expose it as a web service then you are left with one of the following 2 choices together with being WS-I complaint –
   1. RPC/Literal

You cannot use Document/Literal -unwrapped, since more than one child element is not allowed under soap:Body element in SOAP Message by WS-I. The advantage of document/literal-wrapped over RPC/Literal is that the complete message under soap:Body is schema-defined and can be validated with the help of Schema Parsers. In RPC/Literal, the web service operation name and part names are not part of the XML Schema.

5. Advantage of Document/Literal-Wrapped over Document/Literal (Unwrapped)
   Although document/literal-wrapped is a form of document/literal style of messaging, but if the overall document/literal style of messaging is compared with restricted/wrapped document/
literal, the disadvantage with a SOAP Message using document/literal (unwrapped) is that it does not contain details of which operation of the web service corresponds to the incoming XML document. So, if two distinct operations of a Web Service have the same type of method parameter, then dispatching will be difficult for SOAP Engine. The operation information is available with RPC style of messaging. The document literal wrapped design pattern resolves this shortcoming of document/literal style by adding a wrapper element around the document representing the input parameters of the operation.

**Discussion: Overloaded Methods & Wrapped Document/Literal Pattern**

**Method Overloading**

**Listing 4. Overloaded Method "add"**

```java
public void add(int addReq){}
public void add(String addReq){}
```

1. This pattern **CANNOT** be used in cases of method overloading. Method overloading using this pattern would require two elements with the same name in the WSDL-types/schema. However, this is possible through the XML Schema rules (different namespace), but in case of Web Service, method overloading is not encouraged.

2. **WS-I Basic Profile 1.0 (see R2304)** requires operations within a wsdl:portType to be uniquely named. So, from a WS-I compliance perspective, there should be no case of operation overloading.

3. **WSDL 2.0** does not support operation overloading. Hence overloaded operations might not see the light in the future and should be avoided.

**Conclusion**

This article identifies the value addition provided by wrapped document literal pattern in the WSDL design and the shortcoming on method overloading programming practice. But the method overloading has itself been discouraged from the WS-I recommendation and that leaves the designer of “interoperable” web services to unequivocally opt for the wrapped document/literal style of binding when defining the web service.

By virtue of its wide acceptance and high index of interoperability, this pattern also adds value to the close relationship between web services and SOA of enterprise systems, where heterogeneous systems interact with each other and each of bottom-up and top-down design approaches are equally common implementations.
Related topic

• "Which style of WSDL should I use?" (developerWorks, May 2005) describes the WSDL and SOAP messages for each combination to help you decide.