XmnBeansInWsm

This write up discusses the XMLBean integration with Beehive WSM. I have been working on prototypes to address the issues which I should submit soon. The intention of this write up is to explain the current issues requesting comments, suggestion, and alternatives.

The issues discussed here only relates to the Axis integration and is not part of the core WSM.

Problem

There are currently two existing problems that have been documented in JIRA:

http://issues.apache.org/jira/browse/BEEHIVE-843
http://issues.apache.org/jira/browse/BEEHIVE-839

To better understand the problems see: http://xmlbeans.apache.org/docs/2.0.0/guide/conJavaTypesGeneratedFromUserDerived.html

For this discussion I use the term Document, Anonymous schema types and User-Derived types that are explained above.

There roots of the issues are:

a) Axis works well with User-derived types. A document or anonymous types are more than just a type. Additional work has to be done to make a document (or anonymous type) to be used as a type (as viewed by Axis). The current Beehive WSM implementation works with User-Derived types but would have issues with Document and Anonymous types. b) XMLBean objects don't have a method to get their schema types in XML such that it can be added to a WSDL. Each XML Bean object knows the schema file that was used to generate the type. The schema file is treated as source file and the generated classes are viewed as compiled version of the schema. To move the schema to WSDL, we need to read the XML Schema files, parse it and move the required pieces to the WSDL schema. When copying the schema information there are namespace issues that is common for User-Derived types and Documents. Documents (and Annonymous types) however have additional issues that are discussed below.

The goal of my prototype is to solve both of the above issues.

Use cases

For purpose of this discussion I am going to use the following schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified" targetNamespace="http://byXmlBeanNS" xmlns:impl="http://byXmlBeanNS" xmlns:xsd="http://www.w3.org
/2001/XMLSchema">

<xsd:element name="Person">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="name" type="xsd:string"/>
<xsd:element name="addr" type="impl:Address"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:complexType name="Address">
<xsd:sequence>
<xsd:element name="city" nillable="true" type="xsd:string"/>
<xsd:element name="zip" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Compiling this schema with XMLBeans would generate the following classes:

PersonDocument PersonDocument.Person (inner class) Address

.I would like to be able to write methods:

```
@WebMethod void doFoo1(Address myAddress)
@WebMethod void doFoo2(PersonDocument myPerson)
@WebMethod void doFoo3(PersonDocument.Person myPerson)
@WebMethod void doFoo4(AddressHolder myAddressHolder)
@WebMethod void doFoo4(PersonDocumentHolder myPersonHolder)

@WebMethod Address doBar1(Address myAddress)
@WebMethod Address doBar2(PersonDocument myPerson)
@WebMethod Address doBar3(PersonDocument.Person myPerson)
@WebMethod Address doBar5 (AddressHolder myAddressHolder)
@WebMethod Address doBar4(PersonDocumentHolder myPersonHolder)

@WebMethod PersonDocument doFooBar1(Address myAddress)
@WebMethod PersonDocument doFooBar2(PersonDocument myPerson)
@WebMethod PersonDocument doFooBar3(PersonDocument.Person myPerson)
@WebMethod PersonDocument doFooBar5 (AddressHolder myAddressHolder)
@WebMethod PersonDocument doFooBar4(PersonDocumentHolder myPersonHolder)
```

The methods that are shown in Bold would work with the current WSM, the rest would not.

Note that I would like to be able to use both PersonDocument and PersonDocument.Person in my methods. For now I am assuming the returned values from methods, and holders can only be User-Derived types (Address) or Document types (PersonDocument). Anonymous types (at least for now) can't (at least for now) be used in holders or as return values.

When a Document or Anonymous types is used in the method as in:

@WebMethod void doFoo2(PersonDocument myPerson)
@WebMethod void doFoo3(PersonDocument.Person myPerson)

You may want to use the Document as a document or as a type. For instance, assuming a wrapped web service you may want your message body to look:

```
<doFoo2>
<Person>
<name>joe</name>
<addr>
<city>Seattle</city>
<zip>98000</zip>
<addr>
</Person>
</doFoo2>
```

Or

```
<doFoo2>
<myPerson>
<Person>
<name>joe</name>
<addr>
<city>Seattle</city>
<zip>98000</zip>
<addr>
</Person>
</myPerson>
</doFoo2>
```

In the first case documents are used as documents and in the second case a document is just a type. For my current prototype I am assuming the first case. At some point we should allow the user to select the use case (this can be done with @WebParam).

An interesting point to note here is that both methods (doFoo2, doFoo3) would generate the same message and hence identical WSDL, even though the type that is used in the method is different.

Issues

To support the above uses cases there are issues to be dealt with in:

• Service Configuration
• Serialization
• Deserialization
• Invocation
• WSDL Generation

Each of the issues is discussed below.

Service Configuration

For method:

@WebMethod void doFoo2(PersonDocument myPerson)
@WebMethod void doFoo3(PersonDocument.Person myPerson)

Axis expects myPerson to be an element of the type PersonDocument. The problem is that our element is Person (rather than myPerson) and our type is an anonymous type. Similar issue exists when the document type is used as return value:

@WebMethod PersonDocument doFooBar1(Address myAddress)

In this case Axis expects the "return" element to be of the type PersonDocument.

Solution:

My solution to this problem is to define a virtual type "PersonDocumentType" for the element Person, If WSM recognizes a Document or Annonymous type it would configure the service with the element name of the root of the XMLObject and name the type as "xxxDocumentType". So the parameter description would look like:

javaType = xxxDocument, or xxxDocument.Person QName = qname of the root element of the document typeQName = xxxxDocumentType

The "xxxDocumentType" is only used internally; it would not be exposed to the user in WSDL. More on this in WSDL generation issue. The javaType matches the real type of the parameter, more on this in Invocation issue.

Serialization/De-serialization

We would like to be able to use the Document and the Anonymous class in the methods. However, as far as the type system is concern they have the same XML signature in the message and hence we can only use one of them in type mapping configuration. This is particularly issue for deserializer. Once the QName of the type is recognized, the deserailzer needs to find the appropriate deserializer for it to convert the XML to a java type.

Solution

The deserializer would always be configured for the document class with the QName of the root object.

One side effect of having only Document type as de-serializer is that we can only use Document types in Holders. For now this should be an acceptable limitation.

Axis expects the serializer to write the schema of the type it serializes. This wont work for us; the issue is explained further in the WSDL generation section.

The de-serializer must also be intelligent to recognize the difference between User-Derived type and Document types. The user derived types should be de-serialized to XML Fragment where as the Document types should be de-serialized to a XML Document.

Invocation

- Since the serializer is not aware of the actual java type of the method parameters (it always deserializes to document) before invoking a service method we may need to perform the Document to Anonymous type conversion.

Solution

In the Axis architecture Providers are responsible for invoking the methods on the service. In our implementation ControlProvider class is configured as the provider for the service. The name ControlProvider is not a appropriate name of this class (BeehiveProvider or similar name would be more appropriate), nevertheless, the provider can and should perform the argument conversions as needed.

If a method requires the Anonymous class, then provider must introspect the Document class and call the service with the root object of the document as oppose to the document class.

WSDL Generation

WSDL generation has to be discussed separately in case of Bare and Wrapped case.

Let’s assume I have method:

@WebMethod void doFoo2(PersonDocument myPerson)

As far as Axis is concern, if the service for above method is a Bare type, the WSDL schema would look like:

```
<wsdl:types>
<schema elementFormDefault="qualified" targetNamespace="http://byXmlBeanNS" xmlns="http://www.w3.org/2001/XMLSchema">
<element name="Person" type="tns1:PersonDocumentType"/>
</schema>
</wsdl:types>
```

For Wrapped service the WSDL would look like:

```
<wsdl:types>
<schema elementFormDefault="qualified" targetNamespace="http://web" xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://byXmlBeanNS"/>
<element name="doFoo2">
<complexType>
<sequence>
<element name="Person" type="tns1:PersonDocumentType"/>
</sequence>
</complexType>
</element>
</schema>
</wsdl:types>
```

Notice that in case of Bare there are no method over loading, so there can only be one method with <Person> where as the wrapped case the <Person> element may be used in any number of methods, and hence the <element name=”Person” type=”tns1:PersonDocumentType” /> may be used in any number of types.

To the above schema, Axis expects the serializer for the PersonDocumentType to add the schema for the type. This would work fine if the type is the User-Derived type as there is either a simpleType or complexType node in the XML Bean schema of the type. Thus in case of the User-derived types, the serializer can move the schema node form the schema file to the WSDL, but this wont work for the Document and Annonymoyus types.

The problems are:

• The schema file has the element definition for the “Person” and not the type of the Person, we need to find a way to merge the Person element definition with the partial WSDL that Axis has generated
• In case of Bare the element definition is only used once, in case of the Wrapped it is used in multiple places inside complexTypes. Hence the complex types has to be modified to use the schema of the types.

Solution

To generate the correct WSDL we can let Axis generate its partial WSDL then edit the WSDL and add XMLBean types afterward. This wont be possible to do in the serializer (as is expected in Axis). The Provider on the other hand is in position to solve this issue. To generate WSDL, Axis would go through all the types and generate the WSDL as DOM document and save the WSDL on the message context. Axis would then call on the provider to generate WSDL. In my solution the provider then edits the WSDL with the following rules:

a) Parse the DOM node of the WSDL to XMLBean document for the WSDL b) For every “xxx” Document or Annonymous type in the service, remove top level elements such as <element name=”xxx” type=”xxxDocumentType” />.
c) For every “xxx” Document or Annonymous type in the service, change every element that is defined as <element name=”xxx” type=”xxxDocumentType” /> to <element ref=”xxx” />
d) Get the schema file for the “xxx” type, parse it with XMLBeans, and add the schema element to the WSDL (in XMLBeans from (a)) e) Convert the XML Bean WSDL object to DOM node and put it back on the msgContext

Note the above processing would only be needed if the service uses an XMLBean type, otherwise, there is nothing to do for the provider.

(a) is needed because when we move the schema (in step “d”) we need to make sure the namespaces are copied correctly in the new WSDL. In our example the Schema file for Person would have <xsd:element name="addr" type="impl:Address"/> but “impl:Address” doesn’t make sense in the WSDL file, this must be “tns1:Address” as the WSDL file has different namespace mappings.

(b) Is needed to remove the elements that are defined in Bare case. The elements would be copied in step “d”. It would be a NOP in case of Wrapped service.

(c) is needed to fix the wrapped case. Since the schema that is copied (in step “d”) contains the actual “element” definition, the “ref” should be used instead of the element. This would be a NOP in case of Bare service.

(d) is used to add the extra elements for the types

(e) is needed for Axis serialize the wsdl and send it out.

Limitation

The solution that I have proposed would have the following limitations:

• Holders must use Document type.
• Only Document types can be used as method return type.
• In case of Bare, either Document or Anonymous type can be used, it would be error to use both types.

Future Enhancements

• Either fix the above limitation or update the APT process, and model validation to flag the error cases.

• Be able to use XMLObject (base class for XMLBean). Essentiality let the service deal with “any” schema.

• Allow the user to over write the document processing that I outlined here and use a document just as a type. This could be done with @webparam. For instance, If “name” of the parameter is explicitly set to a name that is other than the root of the document, then use the document as a type. To support both behavior we would need to make changes in the service configuration and WSDL generation logic that is described above, this would not be covered in my prototype.