WSDL Java Extension

1 Details

The Java binding is a WSDL binding that allows abstract functionality in the abstract service description (messages, operations and port types) to be mapped to functionality offered by a Java class directly. This means that a Java class can be described using WSDL, and can be accessed as a WSDL-described service through WSIF.

The Java binding extends WSDL with the following extensibility elements:

```
<definitions .... >

    <!-- Java binding -->
    <binding ... >
        <java:binding/>
        <format:typeMapping style="uri" encoding="..."/>?
            <format:typeMap typeName="qname"|elementName="qname"
formatType="nmtoken"/>*
        </format:typeMapping>
        <operation>*
            <java:operation
                methodName="nmtoken"
                parameterOrder="nmtoken"?
                returnPart="nmtoken"?
                methodType="instance|static|constructor"? />?
            <input name="nmtoken"? />?
            <output name="nmtoken"? />?
            <fault name="nmtoken"? />?
        </operation>
    </binding>

    <service ... >
        <port>*
             <java:address
                 className="nmtoken"
                 classPath="nmtoken"?
                 classLoader="nmtoken"? />
        </port>
    </service>

</definitions>
```

Each element is described in detail below.

- **`java:binding`** This indicates that the binding is a Java binding.

- **`format:typeMapping`** For information about the format:typeMapping and format:typeMap please see Format Type Mapping.
- **`java:operation`** This element maps an abstract WSDL operation to a Java method. The methodName attribute specifies the name of the Java method corresponding to the abstract operation. The parameterOrder attribute is similar to and overrides the paramterOrder specification in the abstract operation. It specifies the ordering of the input message parts for the invocation; in the Java binding case it identifies the method signature. Having a parameterOrder attribute here allows us to map an abstract operation to a Java method even if their signatures aren't compatible in the ordering of parts. The returnPart is that part of the abstract output message which corresponds to the return value of the Java method. The methodType attribute specifies whether the Java method is a constructor, a static method or an instance method.
- **`java:address`** This element is an extension under the WSDL port element that allows specification of a Java object as an endpoint for a service available via the Java binding. The port whose address is specified this way must be associated with a Java binding only. The className attribute specifies the fully qualified name of the Java class to be used for service invocation. The optional classPath attribute specifies the classpath to be set prior to invocation, and the optional classLoader attribute specifies the class loader to be used for loading the service class. If invoking an instance method, the service user can load and instantiate the service class; it is up to the service provider to insure that a public no-argument constructor is available. All other Java methods/constructors used for mapping abstract operations must also be publicly available in the service class.

2 Example

```
<?xml version="1.0" ?>

<definitions targetNamespace="http://wsifservice.addressbook/"
             xmlns:tns="http://wsifservice.addressbook/"
             xmlns:typens="http://wsiftypes.addressbook/"
             xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
             xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
             xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- type defs -->
  <types>
    <xsd:schema
      targetNamespace="http://wsiftypes.addressbook/"
                xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <xsd:complexType name="phone">
        <xsd:element name="areaCode" type="xsd:int"/>

        <xsd:element name="exchange" type="xsd:string"/>
        <xsd:element name="number" type="xsd:string"/>
      </xsd:complexType>
      <xsd:complexType name="address">
        <xsd:element name="streetNum" type="xsd:int"/>
        <xsd:element name="streetName" type="xsd:string"/>

        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="state" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:int"/>
        <xsd:element name="phoneNumber" type="typens:phone"/>
      </xsd:complexType>
    </xsd:schema>

  </types>

  <!-- message declns -->
  <message name="AddEntryWholeNameRequestMessage">
    <part name="name" type="xsd:string"/>
    <part name="address" type="typens:address"/>
  </message>

  <message name="AddEntryFirstAndLastNamesRequestMessage">
    <part name="firstName" type="xsd:string"/>
    <part name="lastName" type="xsd:string"/>
    <part name="address" type="typens:address"/>
  </message>

  <message name="GetAddressFromNameRequestMessage">
    <part name="name" type="xsd:string"/>
  </message>

  <message name="GetAddressFromNameResponseMessage">
    <part name="address" type="typens:address"/>
  </message>

  <!-- port type declns -->
  <portType name="AddressBook">
    <operation name="addEntry">
      <input name="AddEntryWholeNameRequest"
message="tns:AddEntryWholeNameRequestMessage"/>
    </operation>
    <operation name="addEntry">

      <input name="AddEntryFirstAndLastNamesRequest"
message="tns:AddEntryFirstAndLastNamesRequestMessage"/>
    </operation>
    <operation name="getAddressFromName">
      <input name="GetAddressFromNameRequest"
message="tns:GetAddressFromNameRequestMessage"/>
      <output name="GetAddressFromNameResponse"
message="tns:GetAddressFromNameResponseMessage"/>
    </operation>

  </portType>
  <!-- binding declns -->
  <binding name="JavaBinding" type="tns:AddressBook">
    <java:binding/>
    <format:typeMapping encoding="Java" style="Java">
      <format:typeMap typeName="typens:address"
formatType="addressbook.wsiftypes.Address" />

      <format:typeMap typeName="xsd:string" formatType="java.lang.String"
/>
    </format:typeMapping>
    <operation name="addEntry">
      <java:operation
         methodName="addEntry"
         parameterOrder="name address"
         methodType="instance" />
      <input name="AddEntryWholeNameRequest"/>
    </operation>

    <operation name="addEntry">
      <java:operation
         methodName="addEntry"
         parameterOrder="firstName lastName address"
         methodType="instance" />
      <input name="AddEntryFirstAndLastNamesRequest"/>
    </operation>
    <operation name="getAddressFromName">
      <java:operation
         methodName="getAddressFromName"
         parameterOrder="name"
         methodType="instance"
         returnPart="address" />

      <input name="GetAddressFromNameRequest"/>
      <output name="GetAddressFromNameResponse"/>
    </operation>
  </binding>

  <!-- service decln -->
  <service name="AddressBookService">

    <port name="JavaPort" binding="tns:JavaBinding">
      <java:address className="addressbook.wsiftypes.AddressBook"/>
    </port>
  </service>

</definitions>
```

In the above example, an address book service is offered through the Java class `addressbook.wsiftypes.AddressBook`. The service offers two variants of the `addEntry` operation. One takes an input message consisting of a full name and an address and the other accepts an input message with a first name, a last name (as separate message parts), and an address. Both of these operations are mapped to a Java method called `addEntry`, but there are different parameter lists (this is an overloaded method in the implementing class). Finally, the abstract operation `getAddressFromName` is mapped to a Java method of the same name. The types `typesns:address` and `xsd:string` used during method invocations are mapped to Java types `addressbook.wsiftypes.Address` and `Java.lang.String` respectively. Note that the type `typesns:phone` does not need to be mapped to a Java type since it is contained within the address type which is represented by a known Java class. WSIF does not need to know how this class represents the information described in the schema type if all that is being done is service invocation. If we needed to inspect the information in the WSIF message and transform it in some way, we would need details on how the Java class represents the type information (see the above discussion on the Java encoding).

All the methods needed for invocation are instance methods, so a client would need to create an instance of the service class. Since no classpath or class loader is specified, the client can assume that there are no special service requirements in this regard, and can just load and instantiate the class and begin using the address book service it implements.