Asif Akram
a.akram@dl.ac.uk

Resource Sharing Among Different Web Services

Till now we have each WSRF service with its own Resource and each client's interaction with Factory Service results in the instantiation of new Resource object through Resource Home class. Is it impressive ….. ? No … can't imagine too much usability and benefits of these state-full resources if we can't extend this concept further. The presentation slides and talks about WSRF and Globus mention sharing of Resources among different WSRF services but never found any proper instructions and tutorials about how to achieve this. There are many scenarios where different Web Services are working on the same resources and exposing different operations corresponding to different ResourceProperty in single ResourcePropertySet, for example Administrator is using different Web Services to manage the users and can modify security credentials, grant different privileges, users are using different Web Service to change only limited set of properties related to their account and then using another Web Service to query information about other users without any ability to modify anything. End of the day different Web Services are working on common resources but have different control on the resources through the operations exposed in the corresponding WSDL. It is not difficult to achieve but conceptually it is not simple to grasp; my solution is working…. and I don't claim this is the best solution.

XML Schema

Come to the implementaion of resource sharing among different Web Services, this is very simple example just to demonstrate the concept. I have three Web Services PersonFactory Service, PersonName Service and PersonAddress Service. PersonFactory Service is only to instantiate the other two services using Factory/Instance design pattern. PersonName and PersonAddress have similar type of ResourcePropertySet which is implemented as PersonResourceHome and PersonResource. PersonName and PersonAddress are using similar type of ResourcePropertySet thus it was better to have complex data types implemented in separate schema rather than in the WSDL, which helps in re-usability of schema, single source of change (which can be single source of error … joking) and uniformity across the application. Below is our simple schema for Person:


```
<xs:schema targetNamespace="http://test.wsrf.resource.sharing"
   elementFormDefault="qualified"
   xmlns="http://test.wsrf.resource.sharing"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:complexType name="name">
      <xs:sequence>
           <xs:element name="title" type="xs:string" />
           <xs:element name="firstName" type="xs:string" />
           <xs:element name="lastName" type="xs:string" />
           <xs:element name="middleName" type="xs:string" />
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="address">
      <xs:sequence>
           <xs:element name="houseNumber" type="xs:string" />
           <xs:element name="streetName" type="xs:string" />
           <xs:element name="city" type="xs:string" />
           <xs:element name="county" type="xs:string" />
           <xs:element name="country" type="xs:string" />
      </xs:sequence>
   </xs:complexType>
```

```
    <xs:complexType name="person">
      <xs:sequence>
          <xs:element name="name" ref="myName" />
          <xs:element name="address" ref="myAddress" />
      </xs:sequence>
    </xs:complexType>

    <xs:element name="myName" type="name" />

    <xs:element name="myAddress" type="address" />

    <xs:element name="myPerson" type="person" />
</xs:schema>
```

Important thing is the target namespace which is http://test.wsrf.resource.sharing thus all these complex data types will be in the package sharing.resource.wsrf.test. Each complex data type is wrapped in the element tag which is requirement of Document/Literal style Web Services as recommended by WS-I Basic Profile, which is another standard to regulate the standards of Web Services.

WSDL for PersonName Service

PersonName Web Service has operations related to the Name part of Person, thus it can retrieves the name of Person and change the name of Person; similarly PersonAdress Web Services has operations to modify and retrieve the address of Person, thus both work on the same resource Person. Below is the WSDL related to PersonName Service.

```
<definitions name="PersonNameService"
    targetNamespace="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:gtwsdl1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsnew="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:myTypes="http://test.wsrf.resource.sharing"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.wsdl"
    location="../wsrf/faults/WS-BaseFaults.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../wsrf/properties/WS-ResourceProperties.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    location="../wsrf/notification/WS-BaseN.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
    location="../wsrf/servicegroup/WS-ServiceGroup.wsdl"/>

  <types>
    <schema
        targetNamespace="http://sharing.wsrf.dl.ac.uk/name/person"
        xmlns:tns="http://sharing.wsrf.dl.ac.uk/name/person"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <import
        namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
```

```
        schemaLocation="../ws/addressing/WS-Addressing.xsd"/>
    <import
        namespace="http://test.wsrf.resource.sharing"
        schemaLocation="Person.xsd"/>

    <import
        namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.xsd"
        schemaLocation="../wsrf/servicegroup/WS-ServiceGroup.xsd"/>

      <element name="PersonResourcePropertiesSet">
        <complexType>
          <sequence>
            <element ref="myTypes:myName"/>
            <element ref="myTypes:myAddress"/>
          </sequence>
        </complexType>
      </element>

      <!-- Elements related to changeName Method -->
      <element name="changeName" ref="myTypes:myName" />
      <element name="changeNameResponse" type="xsd:string" />

      <!-- Elements related to getNameRP Method -->
      <element name="getNameRP" >
        <complexType/>
       </element>
       <element name="getNameRPResponse" ref="myTypes:myName"/>

    </schema>
  </types>

  <message name="ChangeNameRequest">
    <part name="ChangeNameRequest"  element="tns:changeName" />
  </message>
  <message name="ChangeNameResponse">
    <part name="ChangeNameResponse" element="tns:changeNameResponse" />
  </message>

  <message name="GetNameRPRequest">
    <part name="GetNameRPRequest" element="tns:getNameRP" />
  </message>
  <message name="GetNameRPResponse">
    <part name="GetNameRPResponse" element="tns:getNameRPResponse" />
  </message>

  <portType name="PersonNamePortType"
            wsrp:ResourceProperties="PersonResourcePropertiesSet">

    <operation name="GetResourceProperty">
      <input name="GetResourcePropertyRequest"
             message="wsrpw:GetResourcePropertyRequest"
             wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                         ResourceProperties/GetResourceProperty"/>
      <output name="GetResourcePropertyResponse"
              message="wsrpw:GetResourcePropertyResponse"
              wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                         ResourceProperties/GetResourcePropertyResponse"/>
      <fault  name="InvalidResourcePropertyQNameFault"
              message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="ResourceUnknownFault" message="wsrpw:ResourceUnknownFault"/>
    </operation>
```

```
    <operation name="SetResourceProperties">
      <input  name="SetResourcePropertiesRequest"
                   message="wsrpw:SetResourcePropertiesRequest"
                   wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                           ResourceProperties/SetResourceProperties"/>
      <output name="SetResourcePropertiesResponse"
                   message="wsrpw:SetResourcePropertiesResponse"
                   wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                            ResourceProperties/SetResourcePropertiesResponse"/>
      <fault  name="ResourceUnknownFault"
                   message="wsrpw:ResourceUnknownFault" />
      <fault  name="InvalidSetResourcePropertiesRequestContentFault"
                    message="wsrpw:InvalidSetResourcePropertiesRequestContentFault" />
      <fault  name="UnableToModifyResourcePropertyFault"
                   message="wsrpw:UnableToModifyResourcePropertyFault" />
      <fault  name="InvalidResourcePropertyQNameFault"
                   message="wsrpw:InvalidResourcePropertyQNameFault" />
      <fault  name="SetResourcePropertyRequestFailedFault"
                   message="wsrpw:SetResourcePropertyRequestFailedFault" />
    </operation>

    <operation name="Subscribe">
      <input message="wsntw:SubscribeRequest"
          wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification/Subscribe"/>
      <output message="wsntw:SubscribeResponse"
          wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification/SubscribeResponse"/>
      <fault name="TopicPathDialectUnknownFault"
           message="wsntw:TopicPathDialectUnknownFault"/>
      <fault name="SubscribeCreationFailedFault"
           message="wsntw:SubscribeCreationFailedFault"/>
      <fault name="ResourceUnknownFault" message="wsntw:ResourceUnknownFault"/>
    </operation>

    <!--  name attribute in input and output element is optional-->
    <operation name="changeName">
      <input  name="ChangeNameRequest"  message="tns:ChangeNameRequest" />
      <output name="ChangeNameResponse" message="tns:ChangeNameResponse" />
    </operation>

    <operation name="getNameRP">
      <input  name="GetNameRPRequest"  message="tns:GetNameRPRequest" />
      <output name="GetNameRPResponse" message="tns:GetNameRPResponse" />
    </operation>

  </portType>
</definitions>
```

The WSDL file is very simple the only bit tricky things is to import our separate schema file, first we have to declare its namespace in the <definition> section of the WSDL file:

```
xmlns:myTypes=http://test.wsrf.resource.sharing
```

and then import the schema in the <schema> part of the WSDL:

```
<import
     namespace="http://test.wsrf.resource.sharing"
     schemaLocation="Person.xsd"/>
```

and now we can use elements defined in the schema in the WSDL with fully qualified names. The WSDL has two operations changeName and getNameRP which are related to name part of Person type; changeName changes the name of the person and getNameRP retrieves the name of Person, although we

don't need this getNameRP as WSRF operation GetResourceProperty will retrieve both name and address of the person, but this is just to prove the concept.

WSDL for PersonAddress Service

WSDL for PersonAddress Service is very similar to the WSDL for PersonName Service with two business operations changeAddress and getAddressRP. Below is the WSDL file of PersonAddress Service:

```
<definitions name="PersonAddressService"
    targetNamespace="http://sharing.wsrf.dl.ac.uk/address/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/address/person"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:gtwsdl1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:myTypes="http://test.wsrf.resource.sharing"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-BaseFaults-1.2-draft-01.wsdl"
    location="../wsrf/faults/WS-BaseFaults.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../wsrf/properties/WS-ResourceProperties.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.wsdl"
    location="../wsrf/notification/WS-BaseN.wsdl"/>
  <import
    namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.wsdl"
    location="../wsrf/servicegroup/WS-ServiceGroup.wsdl"/>

  <types>
    <schema
      targetNamespace="http://sharing.wsrf.dl.ac.uk/address/person"
      xmlns:tns="http://sharing.wsrf.dl.ac.uk/address/person"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <import
      namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
      schemaLocation="../ws/addressing/WS-Addressing.xsd"/>

    <import
      namespace="http://test.wsrf.resource.sharing"
      schemaLocation="Person.xsd"/>

    <import
      namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-01.xsd"
      schemaLocation="../wsrf/servicegroup/WS-ServiceGroup.xsd"/>

      <element name="PersonResourcePropertiesSet">
        <complexType>
          <sequence>
            <element ref="myTypes:myName"/>
            <element ref="myTypes:myAddress"/>
          </sequence>
        </complexType>
      </element>

      <!-- Elements related to changeAddress Method -->
      <element name="changeAddress" ref="myTypes:myAddress" />
      <element name="changeAddressResponse" type="xsd:string" />
```

```
      <!-- Elements related to getAddressRP Method -->
      <element name="getAddressRP" >
        <complexType/>
       </element>
       <element name="getAddressRPResponse" ref="myTypes:myAddress"/>

    </schema>
  </types>

  <message name="ChangeAddressRequest">
    <part name="ChangeAddressRequest"  element="tns:changeAddress" />
  </message>
  <message name="ChangeAddressResponse">
    <part name="ChangeAddressResponse" element="tns:changeAddressResponse" />
  </message>

  <message name="GetAddressRPRequest">
    <part name="GetAddressRPRequest" element="tns:getAddressRP" />
  </message>
  <message name="GetAddressRPResponse">
    <part name="GetAddressRPResponse" element="tns:getAddressRPResponse" />
  </message>

  <portType name="PersonAddressPortType"
            wsrp:ResourceProperties="PersonResourcePropertiesSet">

    <operation name="GetResourceProperty">
      <input name="GetResourcePropertyRequest"
             message="wsrpw:GetResourcePropertyRequest"
             wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                 ResourceProperties/GetResourceProperty"/>
      <output name="GetResourcePropertyResponse"
              message="wsrpw:GetResourcePropertyResponse"
              wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                  ResourceProperties/GetResourcePropertyResponse"/>
      <fault  name="InvalidResourcePropertyQNameFault"
              message="wsrpw:InvalidResourcePropertyQNameFault"/>
      <fault name="ResourceUnknownFault" message="wsrpw:ResourceUnknownFault"/>
    </operation>

    <operation name="SetResourceProperties">
      <input  name="SetResourcePropertiesRequest"
              message="wsrpw:SetResourcePropertiesRequest"
              wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                          ResourceProperties/SetResourceProperties"/>
      <output name="SetResourcePropertiesResponse"
              message="wsrpw:SetResourcePropertiesResponse"
              wsa:Action="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-
                     ResourceProperties/SetResourcePropertiesResponse"/>
      <fault  name="ResourceUnknownFault"
              message="wsrpw:ResourceUnknownFault" />
      <fault  name="InvalidSetResourcePropertiesRequestContentFault"
              message="wsrpw:InvalidSetResourcePropertiesRequestContentFault" />
      <fault  name="UnableToModifyResourcePropertyFault"
              message="wsrpw:UnableToModifyResourcePropertyFault" />
      <fault  name="InvalidResourcePropertyQNameFault"
              message="wsrpw:InvalidResourcePropertyQNameFault" />
      <fault  name="SetResourcePropertyRequestFailedFault"
              message="wsrpw:SetResourcePropertyRequestFailedFault" />
    </operation>
```

```
    <operation name="Subscribe">
      <input message="wsntw:SubscribeRequest"
       wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification/Subscribe"/>
      <output message="wsntw:SubscribeResponse"
          wsa:Action="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification/SubscribeResponse"/>
      <fault name="TopicPathDialectUnknownFault" message="wsntw:TopicPathDialectUnknownFault"/>
       <fault name="SubscribeCreationFailedFault" message="wsntw:SubscribeCreationFailedFault"/>
      <fault name="ResourceUnknownFault" message="wsntw:ResourceUnknownFault"/>
    </operation>

    <!--  name attribute in input and output element is optional-->

    <operation name="changeAddress">
      <input  name="ChangeAddressRequest"  message="tns:ChangeAddressRequest" />
      <output name="ChangeAddressResponse" message="tns:ChangeAddressResponse" />
    </operation>

    <operation name="getAddressRP">
      <input  name="GetAddressRPRequest"  message="tns:GetAddressRPRequest" />
      <output name="GetAddressRPResponse" message="tns:GetAddressRPResponse" />
    </operation>

  </portType>
</definitions>
```

WSDL for PersonFactory Service

PersonFactory Service has only one “create” method which returns the End Point Reference of the resource, which is similar to previous Factory Services. We need only one Factory Service to instantiate the Resources for both PersonName and PersonAddress services.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PersonFactoryService"
    targetNamespace="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
  <xsd:schema targetNamespace="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:tns="http://sharing.wsrf.dl.ac.uk/name/person"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="../ws/addressing/WS-Addressing.xsd"/>

      <xsd:element name="createResource">
            <xsd:complexType/>
      </xsd:element>
      <xsd:element name="createResourceResponse">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="wsa:EndpointReference"/>
            </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

  </xsd:schema>
</types>
```

```
<message name="CreateResourceRequest">
      <part name="request" element="tns:createResource"/>
</message>
<message name="CreateResourceResponse">
      <part name="response" element="tns:createResourceResponse"/>
</message>

<portType name="PersonNameFactoryPortType">
      <operation name="createResource">
            <input message="tns:CreateResourceRequest"/>
            <output message="tns:CreateResourceResponse"/>
      </operation>
</portType>

</definitions>
```

Implementation Details

In the whole application we have following Java Classes and the table below presents the overall view of the classes and their role before discussing them in details. It is not the implementation of the services which is difficult to grasp, it is their deployment descriptors which are bit tricky.

JAVA Classes	*Their Role in the Application*
PersonFactoryService	There is only one Factory Service which is deployed twice to instantiate both other Services i.e. PersonNameService & PersonAddressService. Technically there is no need to deploy it twice for both services; better approach was to change the CreateResource method which takes String and based on the String it initialize the corresponding service. I preferred to keep the code simple; but later I will discuss the other option also just to complete the topic.
PersonNameService	This Service is simple service only wrapping the operations related to the Name of the Person.
PersonAddressService	Similar to PersonNameService with only difference that this has logic to change the Address property of the Person Resource Property Set.
PersonResourceHome	Our Resource Home class, similar to the previous tutorials and nothing much has changed in it. Be sure we have only one Resource shared among two services therefore we have only one Resource Home also.
PersonResource	Resource to be shared among different services, similar to our previous tutorials with only difference that this time Resource Properties are themselves objects rather than simple Strings, but from implementations point of view nothing makes too difference. Resource class is implemented on the lines outlined in previous tutorials.
PersonQNameService	Interface to create Qualified Names for our Data with static and final varaibles. One important thing to remember, which can be considered as bug, the qualified names declared in this interface only matters and client should use the same qualified names for different Resource Properties accordingly. In other words if WSDL file has Resource Property with local name as “address” and in the interface the same Resource Property has local name “myAddress” even then Server Side implementation will work, but now client has to use “myAddress” rather than “address” to access the Resource Property.

Implementation Details of PersonServiceQNames

PersonServiceQNames is simple interface with final and static variables; which are QName of the ResourceProperties declared in the WSDL. The time only difference in the interface is that we are using two different namespaces declared in the interface, one for our separate schema i.e. http://test.wsrf.resource.sharing and one for the namespace of the services declared in the WSDL i.e. http://test.wsrf.dl.ac.uk/name/person .

Technically these namespaces doesn't make too much difference as they can be anything even different from the what declared in the WSDL and the schema but using different namespaces kill the purpose as then client has to use the same namespace and QNames as used by the server side implementation, and client only guess the QNames of the ResourceProperties from the WSDL. Therefore, namespace and QNames on the server side implementation should match namespace and QNames declared in the WSDL, although WSCore; Globus implementation of WSRF doesn't impose this restriction on the server side implementation.

```
package uk.ac.dl.wsrf.sharing.name.person;

import javax.xml.namespace.QName;

public interface PersonServiceQNames {

  public static final String NS = "http://test.wsrf.dl.ac.uk/name/person";
  public static final String NS2 = "http://test.wsrf.resource.sharing";

  public static final QName RPNAME = new QName(NS2, "myName");
  public static final QName RPAddress = new QName(NS2, "myAddress");
  public static final QName RESOURCEPROPERTIES = new QName(NS,
    "NameResourcePropertiesSet");
}
```

Implementation Details of PersonFactoryService

Implementation of PersonFactoryService is very similar to our previous Factory Services with just single method createResource which creates the EndpointReference for the Resource and returns to the client. In reality Factory Service has nothing to do with instance service, Factory Service only instantiate the Resource which it shares with Instance Service and manually creates the EndpointReference for the instance service and returns.

In deploy-jndi-config.xml for Factory Service we have something similar to the following where Factory Service doesn't have its own resource but has link to the resource declared in "AnyService":

```
<service name="AnyService">
        <resource
            name="home" type="XXX.XXX.XXX">
            <resourceParams>
                ……..
            </resourceParams>
        </resource>
    </service>
<service name="FactoryService">
      <resourceLink name="home" target="java:comp/env/services/AnyService/home"/>
</service>
```

If the name of resource in the "AnyService" is changed to "myResource" then FactoryService will have something like:

```
   <resourceLink name="home" target="java:comp/env/services/AnyService/myResource"/>
```

The following code fragment which we have used again and again in our previous examples will fetch only that resource whose name is “home” in the deploy-jndi-config.xml; if resource has any other name then it can’t be fetched or instantiated directly and the only option is to use InitialContext.

```
ctx = ResourceContext.getResourceContext();
home = (PersonResourceHome) ctx.getResourceHome();
```

Thus FactoryService is only instantiating the resource which it is sharing with the instance service (in above example it is “AnyService”).
Below is the code from PersonFactoryService class:

```
package uk.ac.dl.wsrf.sharing.name.person;

import java.rmi.RemoteException;
import java.net.URL;

import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceKey;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.apache.axis.MessageContext;
import org.globus.wsrf.utils.AddressingUtils;
import org.globus.wsrf.container.ServiceHost;
import uk.ac.dl.wsrf.sharing.name.person.*;

public class PersonFactoryService {

    /* Implementation of createResource Operation */
    public CreateResourceResponse createResource(CreateResource request) throws
            RemoteException {
        System.out.println("PersonFactoryService create");
        ResourceContext ctx = null;
        PersonResourceHome home = null;
        ResourceKey key = null;

        /* First, we create a new PersonResource through the PersonResourceHome  */
        try {
            ctx = ResourceContext.getResourceContext();
            home = (PersonResourceHome) ctx.getResourceHome();
            key = home.create();
        } catch (Exception e) {
            e.printStackTrace();
        }
        EndpointReferenceType epr = null;

        /* We construct the instance's endpoint reference. The instance's service
         * path can be found in the WSDD file as a parameter. */
        try {
            URL baseURL = ServiceHost.getBaseURL();
            String instanceService = (String) MessageContext
                    .getCurrentContext().getService().getOption("instance");
            String instanceURI = baseURL.toString() + instanceService;
            // The endpoint reference includes the instance's URI and the resource key
            epr = AddressingUtils.createEndpointReference(instanceURI, key);
        } catch (Exception e) {
            e.printStackTrace();
        }

        /* Finally, return the endpoint reference in a CreateResourceResponse */
        CreateResourceResponse response = new CreateResourceResponse();
        response.setEndpointReference(epr);
```

```
        return response;
    }
}
```

Implementation of PersonResource

Implementation of PersonResource is very similar to our previous implementations and most of code here is all about Topic, TopicList and Notification.

```
package uk.ac.dl.wsrf.sharing.name.person;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.ResourceIdentifier;
import org.globus.wsrf.impl.SimpleResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;
import org.globus.wsrf.TopicListAccessor;

import uk.ac.dl.wsrf.sharing.name.person.*;
import sharing.resource.wsrf.test.*;
import org.globus.wsrf.TopicList;
import org.globus.wsrf.impl.SimpleTopicList;
import org.globus.wsrf.impl.ResourcePropertyTopic;
import org.globus.wsrf.Topic;

public class PersonResource implements Resource, ResourceProperties,
        ResourceIdentifier, TopicListAccessor {

    /** the identifier of this Resource */
    private Object id;

    /** Stores the ResourceProperties */
    private ResourcePropertySet propSet;

    /* Variable for Resource property */
    private Name name;
    private Address address;


    /** Resource property name and address */
    private ResourceProperty nameRP;
    private ResourceProperty addressRP;


    /** Create Topic List**/
    private TopicList topicList;

    /** initializes the PersonResource. */
    public void initialize() throws Exception {
        System.out.println("PersonResource.initialize() PersonService");

        // choose an ID
        this.id = new Integer(hashCode());

        // create the resource property set
        this.propSet = new SimpleResourcePropertySet(
            PersonServiceQNames.RESOURCEPROPERTIES);
```

```
        // create resource properties
        this.nameRP = new SimpleResourceProperty(PersonServiceQNames.RPNAME);
        this.addressRP = new SimpleResourceProperty(PersonServiceQNames.
                RPAddress);

        //Initialize TopicList
        this.topicList = new SimpleTopicList(this);

        // Wrap ResourceProperty to make it Topic
        this.nameRP = new ResourcePropertyTopic(this.nameRP);
        this.addressRP = new ResourcePropertyTopic(this.addressRP);

        // Add RP in the TopicList
        this.topicList.addTopic((Topic)this.nameRP);
        this.topicList.addTopic((Topic)this.addressRP);

        this.propSet.add(this.nameRP);
        setName(new Name("Asif ", "Akram", " ", "Mr."));
        this.nameRP.add(name);

        this.propSet.add(this.addressRP);
        setAddress(new Address("Daresbury", "United kingdom", "Cheshire", "23",
                               "Wilson Patten"));
        this.addressRP.add(address);

    }

    public ResourcePropertySet getResourcePropertySet() {
        return propSet;
    }

    public void setNameRP(Name nameValue) {
        setName(nameValue);
        this.nameRP.set(0, name);
    }

    public Name getName() {
        return name;
    }

    public void setName(Name name) {
        this.name = name;
    }

    public void setAddressRP(Address addressValue) {
        setAddress(addressValue);
        this.addressRP.set(0, address);
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }


    public Object getID() {
        return id;
    }
```

```
    public TopicList getTopicList() {
        return topicList;
    }
}
```

The important thing is to import sharing.resource.wsrf.test.* where WSDL2Java will create Java mappings for Name and Address data types which are later used in the resource. Following are the key points in the PersonResource:

1. Local variables for Name and Address
```
/* Variable for Resource property */
private Name name;
private Address address;
```
2. Declaring ResourcePropertySet
```
/** Stores the ResourceProperties */
private ResourcePropertySet propSet;
```
3. Declaring ResourceProperty objects for Name and Address
```
/** Resource property name and address */
private ResourceProperty addressRP;
private ResourceProperty nameRP;
```
4. Initializing the ResourcePropertySet
```
this.propSet = new SimpleResourcePropertySet(PersonServiceQNames.RESOURCEPROPERTIES);
```
5. Initializing the ResourceProperty for Name and Address
```
this.nameRP = new SimpleResourceProperty(PersonServiceQNames.RPNAME);
this.addressRP = new SimpleResourceProperty(PersonServiceQNames. RPAddress);
```
6. Initializing Address and Name ResourceProperty after adding them in the ResourcePropertySet
```
/*Initializing Name and adding in the ResourcePropertySet*/
this.propSet.add(this.nameRP);
setName(new Name("Asif ", "Akram", " ", "Mr."));
this.nameRP.add(name);
/*Initializing Name and adding in the ResourcePropertySet*/
this.propSet.add(this.addressRP);
setAddress(new Address("Daresbury", "United kingdom", "Cheshire", "23",
                       "Wilson Patten"));
this.addressRP.add(address);
```
7. get and set methods for Name and Address objects
```
public Address getAddress() { return address; }
public void setAddress(Address address) { this.address = address }

public Name getName() {  return name;  }
public void setName(Name name) {  this.name = name; }
```
8. get method for Name and Address ResourcePropety
```
public void setNameRP(Name nameValue) {
 setName(nameValue);
 this.nameRP.set(0, name);
}
public void setAddressRP(Address addressValue) {
 setAddress(addressValue);
 this.addressRP.set(0, address);
}
```

The only main issue is that while initializing the Name and Address the properties like title, firstName or lastName are not in the same order as they are declared in the schema so you have to check the order of variables in the constructor of the corresponding class. Better way will be use the default constructor to create the object i.e. Name/Address and set different values through set methods.

Implementation of PersonResourceHome

PersonResourceHome has only create method which instantiates the PersonResource and returns the ResourceKey after adding it in the private HashTable of the super class.

```
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.impl.ResourceHomeImpl;
import org.globus.wsrf.impl.SimpleResourceKey;
import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceContext;

public class PersonResourceHome extends ResourceHomeImpl {
    private static java.util.Hashtable resources = new java.util.Hashtable ();

    private int counter=0;
    public ResourceKey create() {
        System.out.println("PersonResourceHomeFactory create() " + counter++);
        try {
            PersonResource resource = (PersonResource) createNewInstance();
            resource.initialize();
            ResourceKey key = new SimpleResourceKey(keyTypeName, resource.getID());
            this.add(key, resource);
            resources.put(key,resource);
            return key;
        } catch (Exception e) {
            System.out.println("Exception when creating PersonResource: ");
            e.printStackTrace();
            return null;
        }
    }
}
```

Implementation of PersonNameService

In actual implementation I have named both Services as PersonService but service which I am calling PersonNameService is in the package "uk.ac.dl.wsrf.sharing.name.person" and the service termed as PersonAddressService is in the package "uk.ac.dl.wsrf.sharing.address.person". I should have named both services differently but preferred to put them in different packages to distinguish them.

```
package uk.ac.dl.wsrf.sharing.name.person;

import java.rmi.RemoteException;
import org.globus.wsrf.ResourceContext;

import uk.ac.dl.wsrf.sharing.name.person.*;
import sharing.resource.wsrf.test.Name;
public class PersonService {

  public PersonService() throws RemoteException {  }

  public String changeName(Name name) throws RemoteException {
        System.out.println("PersonNameService changeName Method Called");
        PersonResource nameResource = getResource();
        nameResource.setNameRP(name);
        return "Name Changed " + name.getFirstName() + " "+name.getLastName() + " !";
    }

    public Name getNameRP(GetNameRP params) throws RemoteException {
        System.out.println("PersonNameService getNameRP Method Called");
        PersonResource personResource = getResource();
        return personResource.getName();
```

```
    }

    public PersonResource getResource() throws RemoteException {
         System.out.println("PersonNameService getResource() Method Called");
        Object resource = null;
        try {
            resource = ResourceContext.getResourceContext().getResource();

        } catch (Exception e) {
            throw new RemoteException("", e);
        }
        PersonResource personResource = (
                PersonResource) resource;
        return personResource;
    }
}
```

Implementation of PersonAddressService

Similar to the implementation of PersonNameService, this service is very simple and exactly on the same line with only differene that it is manipulating Address Resource Property.

```
package uk.ac.dl.wsrf.sharing.address.person;

import java.rmi.RemoteException;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.wsrf.ResourceContext;
import org.globus.wsrf.ResourceKey;
import org.globus.wsrf.utils.AddressingUtils;
import org.apache.axis.MessageContext;

import uk.ac.dl.wsrf.sharing.name.person.*;
import sharing.resource.wsrf.test.*;
import javax.xml.soap.SOAPElement;
import javax.xml.namespace.QName;
import org.globus.wsrf.impl.SimpleResourceKey;

public class PersonService {

    public PersonService() throws RemoteException {
    }

    public String changeAddress(Address address) throws RemoteException {
        System.out.println("PersonService changeAddress Method Called");
        PersonResource resource = getResource();
        resource.setAddressRP(address);
        return "Address Changed " + address.getCity() + " " +
                address.getCountry() + " !";
    }

    public Address getAddressRP(GetAddressRP params) throws RemoteException {
        System.out.println("PersonService getAddressRP Method Called");
        PersonResource resource = getResource();
        return resource.getAddress();
    }

    public PersonResource getResource() throws RemoteException {
        System.out.println("PersonService getResource() Method Called");
        Object resource = null;
        try {
            resource = ResourceContext.getResourceContext().getResource();
```

```
        } catch (Exception e) {
            throw new RemoteException("", e);
        }
        PersonResource personResource = (
                PersonResource) resource;
        return personResource;

    }
}
```

Deployment Descriptors and Build Script

Technically there was nothing new in the implementation and it was just simple implementation just as it was in the previous tutorials but the trick is how to glue them together through Deployment Descriptors. This time Deployment Descriptors are slightly different and once you know what you are tempting to achieve they are simple enough to understand, I will spend more time on the Deployment Descriptors starting from the deploy-server.wsdd.

```
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:aggr="http://mds.globus.org/aggregator/types"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <service name="PersonNameService" provider="Handler" use="literal" style="document">
    <parameter name="providers" value="GetRPProvider SetRPProvider SubscribeProvider"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="activateOnStartup" value="true"/>
    <parameter name="className" value="uk.ac.dl.wsrf.sharing.name.person.PersonService"/>
    <wsdlFile>share/schema/tutorial/PersonNameservice.wsdl</wsdlFile>
  </service>

  <service name="PersonAddressService" provider="Handler" use="literal" style="document">
    <parameter name="providers" value="GetRPProvider SetRPProvider SubscribeProvider"/>
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="activateOnStartup" value="true"/>
   <parameter name="className" value="uk.ac.dl.wsrf.sharing.address.person.PersonService"/>
    <wsdlFile>share/schema/tutorial/PersonAddressservice.wsdl</wsdlFile>
  </service>

  <!-- Factory service -->
  <service name="PersonNameFactoryService" provider="Handler" use="literal" style="document">
     <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
     <parameter name="scope" value="Application"/>
     <parameter name="allowedMethods" value="*"/>
     <parameter name="instance" value="PersonNameService"/>
     <parameter name="className" value="uk.ac.dl.wsrf.sharing.name.person.PersonFactoryService"/>
     <wsdlFile>share/schema/tutorial/FactoryPersonservice.wsdl</wsdlFile>
  </service>

  <service name="PersonAddressFactoryService" provider="Handler" use="literal" style="document">
    <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
    <parameter name="scope" value="Application"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="instance" value="PersonAddressService"/>
    <parameter name="className" value="uk.ac.dl.wsrf.sharing.name.person.PersonFactoryService"/>
    <wsdlFile>share/schema/tutorial/FactoryPersonservice.wsdl</wsdlFile>
```

```
  </service>
</deployment>
```

PersonNameFactoryService and PersonAddressFactoryService both have same implementation class "uk.ac.dl.wsrf.sharing.name.person.PersonFactoryService", in other words same service is deployed with two different names. The reason for this is because both are instantiating different Instance Services using the parameter name "instance" which returns the name of corresponding Instance Service. In our Factory Service we have something similar to:

```
String instanceService = (String) MessageContext.getCurrentContext().getService().getOption("instance");
```

Thus for both PersonNameFactoryService and PersonAddressFactoryService "instance" will return different values, but this limitation can be solved by having two variables "nameInstance" and "addressInstance" but then FactoryService has to parse the create request message to find the target Instance Service and then call corresponding parameter to create EndPointReference. The easier way is to have two method as createNameService and createAddressService which is very childish approach but easy to implement. I prefer the first approach in which there is one Factory Service with one create method and it creates the corresponding resource by parsing the create request message but then it has to use InitialContext to access corresponding resource. I will talk more on this subject of "Best Programming Practices" later or in the next tutorial. Right now keep things simple.

Details of deploy-jndi-config.xml

deploy-jndi-config.xml is quite different from the previous tutorials. In this tutorial we have one resource declared globally rather than local resource. This globally declared resource is consumed by both Instance Services and indirectly by both Factory Services. There is no requirement to have this resource global can be local to anyone of the Instance Service and other Instance Service just has link to it. The first possibility hich I have implemented is like this:


```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
    <global>
        <resource
                name="myGlobalResource" type="uk.ac.dl.wsrf.sharing.name.person.PersonResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>
                    <value>uk.ac.dl.wsrf.sharing.name.person.PersonResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>
                    <value>{http://sharing.wsrf.dl.ac.uk/name/person}PersonKey</value>
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>
    </global>

    <service name="PersonNameService">
            <resourceLink name="home" target="java:comp/env/myGlobalResource" />
    </service>
```

```
    <!-- Factory service -->
    <service name="PersonNameFactoryService">
         <resourceLink name="home" target="java:comp/env/services/PersonNameService/home"/>
    </service>

    <service name="PersonAddressService">
      <resourceLink name="home" target="java:comp/env/myGlobalResource" />
    </service>

    <!-- Factory service -->
    <service name="PersonAddressFactoryService">
      <resourceLink name="home" target="java:comp/env/services/PersonAddressService/home"/>
    </service>

</jndiConfig>
```

Important things to remember Global Resources are references as "`java:comp/env/XX`", we were using the resourceLink earlier also in the Factory Service which means we were sharing the resource between Instance Service and Factory Service and now the only difference is we are sharing the resource among Instance Services . I am sure if this is that confusing …. Second point to remember is that Factory Service can reference the global resource in the same way as Instance Services are doing. In other words:

```
<resourceLink name="home" target="java:comp/env/myGlobalResource" />
```

is equivalent to:

```
<resourceLink name="home" target="java:comp/env/services/PersonAddressService/home"/>
<resourceLink name="home" target="java:comp/env/services/PersonNameService/home"/>
```

It is just matter of your own choice, I have used two different ways only to show different possibilities.

As I said you don't need to have Global Resource and can have one local resource in any of the Instance Service; in that deploy-jndi-config.xml will be like this:

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">

    <service name="PersonNameService">
      <resource
            name="myGlobalResource" type="uk.ac.dl.wsrf.sharing.name.person.PersonResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>
                    <value>uk.ac.dl.wsrf.sharing.name.person.PersonResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>
                    <value>{http://sharing.wsrf.dl.ac.uk/name/person}PersonKey</value>
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>
```

```
    </service>

    <!-- Factory service -->
    <service name="PersonNameFactoryService">
         <resourceLink name="home" target="java:comp/env/services/PersonNameService/home"/>
    </service>

    <service name="PersonAddressService">
      <resourceLink name="home" target="java:comp/env/services/PersonNameService/home"/>
    </service>

    <!-- Factory service -->
    <service name="PersonAddressFactoryService">
      <resourceLink name="home" target="java:comp/env/services/PersonAddressService/home"/>
    </service>

</jndiConfig>
```

The difference is that even PersonAddressService is referencing the resource through PersonNameService:

```
    <service name="PersonNameFactoryService">
         <resourceLink name="home" target="java:comp/env/services/PersonNameService/home"/>
    </service>
```

I prefer to have global resource which keeps things more neat and clean and uniform. Why I don't have resource declared twice in both Instance Services, yes I can do this but that will not work. If we have same resource declared twice locally then each Instance Service will have its own instance of Resource Home and thus resource created by one Instance Service will not be available to other Instance Service. In other words PersonNameService creates object of PersonResource called resourceObject1, now when you will try to access this resourceObject1 through PersonAddressService you will get the exception no such resource available, because PersonAddressService will try to find resourceObject1 in its own instance of PersonResourceHome. I had the same problem which I initially solved by using static HashTable of resources in the PersonResourceHome where I was adding the resources myself rather than adding in the HashTable of super class and then was locating the resource myself, this approach works but no reason for this extra effort unless you want to do something extra, like monitoring the resources, number of resources for statistical reasons as from PersonResourceHome you can locate/find any specific resource but can't list all existing resources. You may need this option imagine you have Portal where user logins and then you list all the instances created by one specific user or group of users and they click on any specific Resource and you dynamically creates its EPR and update the other part of Portal with its Properties.

Details of Build Script

The last missing part of the puzzle is Build script file. Simple just like other build scripts with only difference that this time we have three WSDL's so need extra targets. Below is the whole build script highlighting the changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!- Name of the project will take the name of the gar file
    Similar directory can be found in the gt-install/etc-->
<project basedir="." default="all" name="globuspersonsharing">

    <property environment="env"/>

    <property file="build.propertPu101 ?s"/>
    <property file="${user.home}/build.properties"/>
      <!- I have removed this as don't need it in Windows was here
           from Sticky Note Tutorial
    <property location="../insstall" name="env.GLOBUSLOCATION"/>
```

```
-->
<property location="${env.GLOBUSLOCATION}" name="deploy.dir"/>
<!—No Idean where person-sharing is used but have for sake of having it-->
<property name="base.name" value="perosnsharing"/>
<property name="package.namÉ" value="globus${base.name}"/>
<property name="jar.name" value="${package.name}.jar"/>
<property name="gar.name" value="${package.name}.gar"/>
<property location="build" name="build.dir"/>
<property location="build/classç" name="build.dest"/>
<property location="build/lib" name="build.lib.Ðir"/>
<property location="build/stubs" name="stubs.dir"/>
<property location="build/stubs/rc" name="stubs.src"/>
<property location="build/stubs/classes" name="stubs.dest"/>
<property name="stubs.jar.name" value="${package.name}stubs.jar"/>
<property location="${deploy.dir}/share/globuswsrfcommon/build-packages.xml" name="build.packages"/>
<property location="${deploy.dir}/share/globuswsrftools/build-stubs.xml" name="build.stubs"/>
<property name="java.debug" value="on"/>

<property name="test-reports.dir" value="test-reports"/>
<property name="junit.haltonfailure" value="true"/>

<property location="${deploy.dir}/share/schema" name="schema.src"/>
<property location="${build.dir}/schema" name="schema.dest"/>

<property name="garjars.id" value="garjars"/>
<fileset dir="${build.lib.du114 ?}" id="garjars"/>

<property name="garschema.id" value="garschema"/>
<fileset dir="${schema.dest}" id="garschema" includes="tutorial/*"/>

<property name="garetc.id" value="garetc"/>
<fileset dir="etc" id="garetc"/>

<target name="init">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${build.dest}"/>
    <mkdir dir="${build.lib.dPu114 ?}"/>

    <mkdir dir="${stubs.dir}"/>
    <mkdir dir="${stubs.src}"/>
    <mkdir dir="${stubs.dest}"/>

    <mkdir dir="${schema.dest}"/>

    <copy toDir="${schema.dest}">
       <fileset dir="${schema.src}" casesensitive="yes">
           <include name="wsrf/**/*"/>
           <include name="ws/**/*"/>
       </fileset>
    </copy>
    <copy toDir="${schema.dest}">
       <fileset dir="schema/" casesensitive="yes">
           <include name="tutorial/*"/>
       </fileset>
    </copy>
  <available file="${stubs.dest}/org.globus.wsrf.mds.aggregator" property="stubs.present" type="dir"/>
</target>

<target name="bindings1" depends="init">
    <ant antfile="${build.stub}" target="generateBindinÐ">
       <property name="source.binÐing.dir"
           value="${schema.dest}/tutorial"/>
```

```
            <property name="target.bin€u105 ?ng.dir"
                value="${schema.dest}/tutorial"/>
            <property name="binding.root" value="PersonName"/>
            <property name="porttype.wsdl" value="PersonName.wsdl"/>
        </ant>
    </target>

    <target name="bindings2" depends="bindings1">
        <ant antfile="${build.stub}" target="generateBindinø">
            <property name="source.bin€u105 ?ng.dir"
                value="${schema.dest}/tutorial"/>
            <property name="target.bin€u105 ?ng.dir"
                value="${schema.dest}/tutorial"/>
            <property name="binding.root" value="PersonAddress"/>
            <property name="porttype.wsdl" value="PersonAddress.wsdl"/>
        </ant>
    </target>

    <target name="bindingsFactory" depends="bindings2">
       <ant antfile="${build.stub}" target="generateBindin8">
            <property name="source.binxing.dir"
                value="${schema.dest}/tutorial"/>
            <property name="target.bin€u105 ?ng.dir"
                value="${schema.dest}/tutorial"/>
            <property name="binding.root" value="FactoryPerson"/>
            <property name="porttype.wsdl" value="FactoryPerson.wsdl"/>
        </ant>

    </target>

    <target name="stubs1" unless="stubs.present" depends="bindingsFactory">
        <ant antfile="${build.stub}" target="generateStub">
            <property location="${schema.dest}/tutorial" name="source.stubs.dir"/>
            <property name="wsdl.file" value="PersonNameservice.wsdl"/>
            <property location="${stubs.src}" name="target.stubs.dir"/>
        </ant>
    </target>

    <target name="stubs2" unless="stubs.present" depends="stubs1">
        <ant antfile="${build.stub}" target="generateStub">
            <property location="${schema.dest}/tutorial" name="source.stubs.dir"/>
            <property name="wsdl.file" value="PersonAddressservice.wsdl"/>
            <property location="${stubs.src}" name="target.stubs.dir"/>
        </ant>
    </target>

    <target name="stubsFactory" unless="stubs.present" depends="stubs2">
        <ant antfile="${build.stub}" target="generateStub">
            <property location="${schema.dest}/tutorial" name="source.stubs.dir"/>
            <property name="wsdl.file" value="FactoryPersonservice.wsdl"/>
            <property location="${stubs.src}" name="target.stubs.dir"/>
        </ant>
    </target>

    <target depends="stubsFactory" name="compileStubs">
        <delete dir="${stubs.src}/org/apache"/>
        <javac debug="${java.debug}" destdir="${stubs.dest}" srcdir="${stubs.src}">
            <include name="**/*.java"/>
            <classpath>
                <fileset dir="${deploy.dir}/lib">
                    <include name="*.jar"/>
```

```
            </fileset>
        </classpath>
    </javac>
</target>



<target depends="compileStubs" name="compile">
    <javac debug="${java.debug}" destdir="${build.dest}" srcdir="src">
        <include name="**/*.java"/>
        <classpath>
            <pathelement location="${stubs.dest}"/>
            <fileset dir="${deploy.dir}/lib">
                <include name="*.jar"/>
            </fileset>
        </classpath>
    </javac>
</target>

<target depends="compileStubs" name="jarStubs">
    <jar basedir="${stubs.dest}" destfile="${build.lib.du114
?}/${stubs.jar.name}"/>
</target>


<target depends="compile" name="jar">
    <jar basedir="${build.dest}" destfile="${build.lib.du114 ?}/${jar.name}"/>
</target>

<target depends="jar, jarStub" name="dist">
    <ant antfile="${build.packages}" target="makeGar">
        <property name="gar.name" value="${build.lib.du114 ?}/${gar.name}"/>
        <reference refid="${garjars.id}"/>
        <reference refid="${garschema.id}"/>
        <reference refid="${garetc.ix}"/>
    </ant>
</target>

<target name="clean">
    <delete dir="tmp"/>
    <delete dir="${build.dir}"/>
    <delete file="${gar.name}"/>
    <delete dir="${test-reports.dir}"/>
</target>

<target depends="dist" name="deploy">
    <ant antfile="${build.packages}" target="deployGar">
        <property name="gar.name" value="${build.lib.du114 ?}/${gar.name}"/>
        <property name="gar.id" value="${package.name}"/>
        <!-- <property name="noSchema" value="true"/> -->
    </ant>
</target>

<target name="undeploy">
    <ant antfile="${build.packages}" target="undeployGar">
        <property name="gar.id" value="${package.name}"/>
    </ant>
</target>

<target depends="deploy" name="all"/>
```

```
    <target depends="compile" name="test">
        <mkdir dir="${test-reports.dir}"/>
        <junit fork="yes" haltonfailure="${junit.haltonfailure}" printsummary="yes" timeout="600000">
           <classpath>
               <pathelement location="${build.dest}"/>
               <pathelement location="${deploy.dir}"/>
               <pathelement location="${deploy.dir}/etc/wsrf-bundles.properties"/>
               <fileset dir="${deploy.dir}/lib">
                   <include name="*.jar"/>
               </fileset>
           </classpath>
           <formatter type="xml"/>
           <batchtest todir="${test-reports.dir}">
               <fileset dir="${build.dest}">
                   <include name="**/*Test.clas"/>
               </fileset>
           </batchtest>
        </junit>
    </target>
</project>
```

Changes are in Bold, Italics and Underlined and nothing much to discuss them.

Details about Client

Everything done, now you are ready to deploy the file, but we need the client to test it. Below is the whole implementation of the client:

```
package uk.ac.dl.wsrf.sharing.name.person.client;

import org.oasis.wsrf.properties.GetResourcePropertyResponse;

import org.apache.axis.message.addressing.EndpointReferenceType;

import org.apache.commons.cli.ParseException;
import org.apache.commons.cli.CommandLine;

import org.globus.wsrf.client.BaseClient;
import org.globus.wsrf.utils.AnyHelper;
import org.globus.wsrf.utils.FaultHelper;

import uk.ac.dl.wsrf.sharing.name.person.PersonServiceQNames;
import uk.ac.dl.wsrf.sharing.name.person.service.*;
import uk.ac.dl.wsrf.sharing.address.person.service.*;
import uk.ac.dl.wsrf.sharing.name.person.*;
import uk.ac.dl.wsrf.sharing.address.person.*;
import sharing.resource.wsrf.test.*;
import javax.xml.rpc.*;

public class ClientName extends BaseClient {

    final static PersonNameServiceAddressingLocator locator =
            new PPersonNameServiceAddressingLocator();

    final static PersonFactoryServiceAddressingLocator factoryLocator =
            new PersonFactoryServiceAddressingLocator();

    final static PersonAddressServiceAddressing locator2 =
            new PersonAddressServiceAddressingLocator();
```

```
    private static EndpointReferenceType service1InstanceEPR = null;
    private static EndpointReferenceType service2InstanceEPR = null;

    private EndpointReferenceType getInstanceEPR(String path) {
        EndpointReferenceType instanceEPR;
        try {
            this.getEPR().getAddress().setPath(path);
            PersonNameFactoryPortType factoryPort = factoryLocator.
                    getPersonNameFactoryPortTypePort(
                            this.getEPR());
            CreateResourceResponse createResponse = factoryPort
                    .createResource(new CreateResource());
            instanceEPR = createResponse.getEndpointReference();
            instanceEPR.getAddress().setPort(8082);
            return instanceEPR;

        } catch (ServiceException ex) {
            ex.printStackTrace();
        } catch (Exception ex1) {
            ex1.printStackTrace();
        }

        return null;
    }

    public static void main(String[] args) {
        ClientName client = new ClientName();

        // first, parse the commandline
        try {
            CommandLine line = client.parse(args);
        } catch (ParseException e) {
            System.err.println("Parsing failed: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            System.exit(1);
        }

        try {
            service1InstanceEPR = client.getInstanceEPR(
                    "/wsrf/services/PersonNameFactoryService");
            /*
            service2InstanceEPR = client.getInstanceEPR(
                    "/wsrf/services/PersonAddressFactoryService");*/
            service2InstanceEPR = new EndpointReferenceType();
            service2InstanceEPR.setAddress(new org.apache.axis.message.
                                    addressing.Address(
                    "http://XX.XX.XX.XX:8082/wsrf/services/PersonService2"));
            service2InstanceEPR.setProperties(service1InstanceEPR.getProperties());
            System.out.println(service2InstanceEPR.getProperties().toString());
            System.out.println("  " + service2InstanceEPR.toString());
            PersonNamePortType port;
            PersonAddressPortType port2;

            // Get instance PortType
            port = locator.getPersonNamePortTypePort(service1InstanceEPR);
            port2 = locator2.getPersonAddressPortTypePort(service2InstanceEPR);
            System.out.println("Created instance.");

            GetResourcePropertyResponse response =
```

```
                port.getResourceProperty(PersonServiceQNames.RP_NAME);
            System.out.println(AnyHelper.toSingleString(response));
            response = port.getResourceProperty(PersonServiceQNames.RP_Address);
            System.out.println(AnyHelper.toSingleString(response));

            Thread.sleep(1000);
            String result = port.changeName(new Name("AsIf", "AkRaM", "…", " Mr."));
            System.out.println(result);
            result = port2.changeAddress(new Address("London", "United Kingdom",
                  "London", "38", "Hamilton Street"));
            System.out.println(result);

            response = port.getResourceProperty(PersonServiceQNames.RP_NAME);
            System.out.println(AnyHelper.toSingleString(response));
            response = port.getResourceProperty(PersonServiceQNames.RP_Address);
            System.out.println(AnyHelper.toSingleString(response));

        } catch (Exception e) {
            if (client.isDebugMode()) {
                FaultHelper.printStackTrace(e);
            } else {
                System.err.println("Error: " + FaultHelper.getMessage(e));
            }
        }
    }
}
```

I have highlighted the code and will discuss one by one. In fact it was the client who needs bit clever logic as compared to our dumb services. I have added the method getInstanceEPR; which takes the String which is the name and location of the Factory Service and calls the corresponding Factory Services. We are calling getInstanceEPR method only once and this really doesn't makes any difference if I call PersonNameFactory or PersonAddressFactory end of the day both are initializing the same resource i.e. PersonResource.

```
private EndpointReferenceType getInstanceEPR(String path) {
        EndpointReferenceType instanceEPR;
        try {
            this.getEPR().getAddress().setPath(path);
            PersonNameFactoryPortType factoryPort = factoryLocator.
                    getPersonNameFactoryPortTypePort(
                            this.getEPR());
            CreateResourceResponse createResponse = factoryPort
                    .createResource(new CreateResource());
            instanceEPR = createResponse.getEndpointReference();
            // We are changing the port also to use TCP Monitor
            instanceEPR.getAddress().setPort(8082);
            return instanceEPR;

        } catch (ServiceException ex) {
            ex.printStackTrace();
        } catch (Exception ex1) {
            ex1.printStackTrace();
        }

        return null;
    }
```

In main method I am calling getInstanceEPR only once by passing the corresponding String, you don't need to call both Factory Services as this will result in the two different objects of PersonResource but you can

call anyone of the Instance Service. One call to the getInstanceEPR is in the comments just for your convenience

```
service1InstanceEPR = client.getInstanceEPR(
        "/wsrf/services/PersonNameFactoryService");
/*
service2InstanceEPR = client.getInstanceEPR(
      "/wsrf/services/PersonAddressFactoryService");*/
```

If I am calling getInstanceEPR then I need to create the EPR for the other Instance Service manually. I am using bits and pieces from the first EPR got from the method call and just changing the name of the instance as both Instance Services will work on the same PersonResource object.

```
service2InstanceEPR = new EndpointReferenceType();
service2InstanceEPR.setAddress(new org.apache.axis.message. addressing.Address(
                "http://XX.XX.XX.XX:8082/wsrf/services/PersonAddressService"));
service2InstanceEPR.setProperties(service1InstanceEPR.getProperties());
```

Replace the XX.XX.XX.XX with the IP of your own computer or use localhost. Third statement is copying the resourceProperties of EPR for first Instance Service to the EPR of other Instance Service which will be something like

```
<ns1:PersonKey xmlns:ns1="http://sharing.wsrf.dl.ac.uk/name/person" soapenv:mustUnderstand="0">17872448</ns1:PersonKey>
```

Later I am calling resource through port and port2 and changing the name and address of the resource. "port" is for PersonNameService and port2 is for PersonAddressService. "changeName" and "changeAddress" methods return the String to confirm the successful execution.

```
String result = port.changeName(new Name("AsIf", "AkRaM", "…", " Mr."));
result = port2.changeAddress(new Address("London", "United Kingdom",
                "London", "38", "Hamilton Street"));
```

My name hasn't changed during this tutorial so I have used mixture of uppercase and lowercase to differentiate it. Initial values are "Mr. Asif Akram" and changed values are "AsIf AkRaM". I have changed my address but it should be other way around as I moved here from London but who knows I may move back to London. If you are wondering about Daresbury then:

"Daresbury Village is a charming place, and it's well worth spending a few hours here just wandering around. The village it situated in the cheshire countryside and the birth place of ***Lewis Caroll*** *who wrote the famous childrens book* ***'Alice's adventures in wonderland'".***

Discussing the Outcome

After running the client you will get the following result which is in accordance to what we are expecting. To run the client use the following command:

%GLOBUS_LOCATION%\bin\ClientPerson -s http://localhost:8082/wsrf/services/PersonNameFactoryService

Changing it from PersonNameFactoryService to PersonAddressFactoryService will not make any difference as we are creating EPR manually in the client.

```
<ns1:myName xmlns:ns1="http://test.wsrf.resource.sharing">
      <ns1:title>Mr.</ns1:title>
      <ns1:firstName>Asif </ns1:firstName>
      <ns1:lastName>Akram</ns1:lastName>
```

```
        <ns1:middleName> </ns1:middleName>
</ns1:myName>
<ns1:myAddress xmlns:ns1="http://test.wsrf.resource.sharing">
        <ns1:houseNumber>23</ns1:houseNumber>
        <ns1:streetName>Wilson Patten</ns1:streetName>
        <ns1:city>Daresbury</ns1:city>
        <ns1:county>Cheshire</ns1:county>
        <ns1:country>United kingdom</ns1:country>
</ns1:myAddress>
Name Changed AsIf AkRaM !
Address Changed London United Kingdom !
<ns1:myName xmlns:ns1="http://test.wsrf.resource.sharing">
        <ns1:title> Mr.</ns1:title>
        <ns1:firstName>AsIf</ns1:firstName>
        <ns1:lastName>AkRaM</ns1:lastName>
        <ns1:middleName>&#x2026;</ns1:middleName>
</ns1:myName>
<ns1:myAddress xmlns:ns1="http://test.wsrf.resource.sharing">
        <ns1:houseNumber>38</ns1:houseNumber>
        <ns1:streetName>Hamilton Street</ns1:streetName>
        <ns1:city>London</ns1:city>
        <ns1:county>London</ns1:county>
        <ns1:country>United Kingdom</ns1:country>
</ns1:myAddress>
```

NOTE:

I have changed the name of implementation classes, services and variables in the tutorial just to make it easy to understand, but in real implementation they are slightly different. Don't copy the stuff from here to run it, you should modify the supplied code according to your requirements.