Grid Business Process: Case Study

Asif Akram[1], Sanjay Chaudhary[2], Prateek Jain[2], Zakir Laliwala[2] and Rob Allan[1]
[1]CCLRC Daresbury Laboratory, Daresbury, UK
{a.akram, r.j.allan@dl.ac.uk @dl.ac.uk
[2]Dhirubhai Ambani Institute of Information and Communication Technology,
Gandhinagar 382 007, Gujarat, India.
{ sanjay_chaudhary, prateek_jain, zakir_laliwala }@daiict.ac.in

Note: In this file, only the implementation of "Grid Business Process..." is discussed.

8. Implementation

The dynamic nature of the agro-marketing case study and requirements to monitor the state of different resources during the lifecycle of business process, are difficult to manage through vanilla Web services. Different built in features of the WS-RF family specifications naturally maps the demanding requirements of the case study and are the first choice for the implementation. Different commercial and open source implementation of the WS-RF specifications are available, and selecting any one of them is more or less personal choice. During this development Globus Toolkit 4.0.1 (GT4) on Red Hat Linux 9 is used to develop and deploy different Web services. Globus implementation of WS-RF comes with embedded container (i.e. Tomcat) and has extensive support for different level of security. The working details of the GT4 are very specific and thorough understandings of them is very crucial to develop any real world application. GT4 is based on the Axis SOAP engine and makes maximum use of different tools and feature available in the Axis toolkit. Before discussing the implementation details of our case study, we have outlined the steps and configuration files involved in the stateful Web services development for the GT4.

8.1 Designing WSDL Document

The first step in the development of the stateful Web Services is to design the WSDL document for the service. This WSDL document should be in compliance with the WS-I Basic Profile recommendations for the Document/literal style services. For the document style WSDL each complex data types are wrapped in the element. Only elements can be referenced from other section of the WSDL documents particularly <wsdl:message> section. GT4 doesn't require the complete WSDL document; the sections required in the WSDL are only related to the interface of the service. GT4 provides the binding and service details of the service at the run time. The WSDL document for the stateful Web Services normally have a complex data type in the <wsdl:type> section, which is not referenced by any operation but it is coupled with the Web Service through the attribute ResourceProperties of <wsdl:portType> element. The attribute ResourceProperties is the extension of the WSDL specification and is only meant for WS-RF compliant engines. Below is the example of WSDL fragment for Buyer service:

```
<xsd:element name="BuyerResourceProperties">
<xsd:complexType>
   <xsd:sequence>
     …….
   </xsd:sequence>
</xsd:complexType>
</xsd:element>
  …….
<portType name="BuyerPortType"
wsrp:ResourceProperties="tns:BuyerResourceProperties">
……..
```

Listing 1. Buyer Service WSDL fragment

8.2 Stub and Proxy Generation

The second step is to generate stubs and proxies from the WSDL document. The Axis framework provides a tool "WSDL2Java", which parses the WSDL and generates appropriate classes compliant with the WSDL interface. The complex data types in the WSDL for Document/literal style Web Services are wrapped in elements. Elements wrapping complex data type are normally mapped to separate Java classes to shield developers from the low level details of XML. During this stubs and proxy generations the package structure is normally derived from the target namespace unless it is specifically mentioned through separate configuration file (i.e. namespace2mapping). Below is the example of automatic mapping of the target namespace into java package structure:

```
targetNamespace="http://manipulation.any.wsrf.dl.ac.uk/addressbook"
maps to "uk.ac.dl.wsrf.any.manipulation.addressbook"
```

Listing 2. Mapping of target namespace to java package

8.3 Deployment Descriptor

The deployment descriptor provides enough information to the Axis framework to successfully deploy the Web service. The normal information required by the Axis engine is the name of the Web service, style of the Web service (it should be Document/literal style), the class implementing the business logic of the Web service, the location and name of the WSDL file and any number of parameters required for the successful instantiation of the service. Below is the example of a Web service Deployment Descriptor which is normally referenced as "wsdd".

```
<service name="AddressBookService" provider="Handler" use="literal"
style="document">
    <parameter name="providers" value="GetRPProvider SetRPProvider GetMRPProvider"/>
    <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="instance" value="ZZZ"/>
    <parameter name="className"
     value="uk.ac.dl.wsrf.addressbook.AddressBookService"/>
    <wsdlFile>share/schema/tutorial/AddressBookManipulation_service.wsdl</wsdlFile>
  </service>
```

Listing 3. Sample Web service Deployment Descriptor

In the example above the Web service is provided with one application specific parameter called "instance" whose value is "ZZZ". The service can access these parameters at run time by using either the JNDI API or the message context. GT4 specific stateful Web services use these parameters very effectively and thorough understanding of them is very important. Below is the java code to retrieve the value of the "instance" parameter.

```
MessageContext.getCurrentContext().getService().getOption("instance");
```

8.4 JNDI Configuration File

Till now we haven't talked too much about the resources coupled with the Web service; except just declaring the data type of the resources in the <wsdl:type> section and then referencing it in the <wsdl:portType> through the "ResourceProperties" attribute. According to the specifications the Web service is independent of the resource itself and it is the responsibility of the container and framework to load appropriate resource and carry out requested operation on the resource. This is one of the reasons, why "wsdd" never mentioned anything about the resource coupled with the service.

The all information required by the container for proper instantiation of the resource is provided in the "deploy-jndi-config.xml". Below is the sample "deploy-jndi-config.xml" file followed by the key points related to the WS-RF based services.

```
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
    <service name="AddressBookService">
        <resource
            name="home" type="uk.ac.dl.wsrf.addressbook.AddressBookResourceHome">
            <resourceParams>
                <parameter>
                    <name>factory</name>
                    <value>org.globus.wsrf.jndi.BeanFactory</value>
                </parameter>
                <parameter>
                    <name>resourceClass</name>
                    <value>uk.ac.dl.wsrf.addressbook.AddressBookResource</value>
                </parameter>
                <parameter>
                    <name>resourceKeyName</name>
                    <value>{http://wsrf.dl.ac.uk/addressbook}AddressBookKey</value>
                </parameter>
                <parameter>
                    <name>resourceKeyType</name>
                    <value>java.lang.Integer</value>
                </parameter>
            </resourceParams>
        </resource>
    </service>

    <!-- Factory service -->
    <service name="AddressBookFactoryService">
   <resourceLink name="AddressBookHome"
 target="java:comp/env/services/AddressBookService/home"/>
    </service>
</jndiConfig>
```

Listing 4. Sample deploy-jndi-config.xml file

The JNDI configuration file is basic file required by the Tomcat container to couple different resources (i.e. data objects, JDBC connection etc) within the application. The service "AddressBookService" has one resource attached to it. For successful coupling of any resource with the service, requires the class for the instantiation of the resource which is normally called ResourceHome, and the class which implements the logic of the resource. The resource can only be instantiated through the resource home. The only information specific to the WS-RF implementation is the resourceKeyName and resourceKeyType. This information is used to search specific type of resource or particular resource within the same type of resources. The resourceKeyName is the qualified name and should map the qualified name of the resource as declared in the corresponding WSDL document.

8.5 Implied Resource Pattern

As discussed in the section 4.1, in the Implied Resource Pattern the Factory service instantiate the appropriate resource and returns the EPR of the resource to the client. The EPR includes the unique ID of the resource and the URL of the Instance service managing the resource.

8.5.1 Factory Service

The role of the Factory service is to instantiate the appropriate resource through its resource home. The Factory service retrieves the information related to the resource and its resource home through the JNDI configuration file. In the JNDI configuration file discussed in the previous section, the Factory service doesn't have any resource associated to it but it do have a link to the resource home for the AddressBookService. The AddressBookFactoryService will retrieve the information about the resource which in fact it is sharing with the AddressBookService, and

instantiate it appropriately. Below is the java code to retrieve the resource home and returning the resulting EPR to the client:

```
try {
            Context initialContext = new InitialContext();
            ctx = ResourceContext.getResourceContext();
            String name = Constants.JNDI_SERVICES_BASE_NAME + ctx.getService() +
                          "/AddressBookHome";
            home = (AddressBookResourceHome) initialContext.lookup(name);
            key = home.create();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Listing 5. Code snippet of Factory Service to retrieve Resource Home

"/AddressBookHome" is the name of the resource link declared in the Factory service. The above code retrieves the resource home object and calls the create() method of the resource home object. The create() method instantiate the resource and returns the unique ID of the resource. The normal practice is to keep the resource instantiation code in the private utility method of the Factory service and this convention is followed during the implementation of the case study. This unique ID is used to create EPR, which is returned to the client. Below is the remaining code which creates the EPR of the resource:

```
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);
            response.setEndpointReference(epr);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Listing 6. Code snippet for EPR creation of resource

The String parameter "instance" is declared in the "wsdd" which deploys the Factory service.

8.5.2 Resource Home

The resource home is simple class with only one create (); which initialize the corresponding resource. The resource home normally extends the GT4 specific ResourceHomeImpl class. The ResourceHomeImpl is very useful class which shields user from lot of implementation. This supper class locates the implementation of the resource through the JNDI configuration file and instantiates it appropriately. The ResourceHomeImpl also provides the private hash table to store all the resources created through the custom tailored resource home. The business logic of searching and updating the private hash table is automatically available in the custom resource home by extending the ResourceHomeImpl. Below is the code for the sample resource home.

```
public class AddressBookResourceHome extends ResourceHomeImpl {
  private String instanceServicePath;
  public ResourceKey create(){
  ResourceKey key=null;
  try{
    // Retrieves the resource class from the JNDI configuration file
    AddressBookResource addressBookResource = (AddressBookResource)
this.createNewInstance();
    addressBookResource.initialize();
    key=new SimpleResourceKey(this.getKeyTypeName(),addressBookResource.getID());
    this.add(key, addressBookResource);
  }  catch (Exception e)    {
     e.printStackTrace();
  }
}
```

Listing 7. Code snippet of AddressBookResourceHome

The most important bit in the method is to create the instance of the resource through createNewInstance() method. This method is implemented in the super class i.e. ResourceHomeImpl; which returns an object instance of respective resource by retrieving the implementation details of the resource from JNDI configuration file. Other than instantiating the resource, the resource home creates the instance of the SuperResourceKey. The SuperResourceKey couples the unique ID of the resource with its fully qualified name. This fully qualified unique ID is used in the EPR. At anytime container can be managing many different types of resources which may have the same unique ID; it is the fully qualified name and the unique ID of the resource which makes any specific instance of the resource unique from other instances.

```
key=new SimpleResourceKey(this.getKeyTypeName(),addressBookResource.getID());
this.add(key, addressBookResource);
```

Listing 8. Coupling of unique resource ID and resource

The implementation details of each resource home developed during the course of the case study are very similar and it will not be discussed further with actual implementation details of that Web service.

8.6 Grid Registry

The role of the Grid Registry is to monitors all resources instantiated during the life cycle of the application; irrespective of its nature and type. The resource home only monitors and manage resources of specific type, so they can't be used directly as Grid Registry. Although it is possible to develop Grid Registry on the top of all different resource homes; but this requires lot of effort to implement the business logic of searching, querying and modifying these individual resources. The GT4 implementation provides the DefaultIndexService built according to the specifications of WS-ServiceGroup discussed in the section 3.2.3. This Index Service maps to our concept of the Grid Registry; which aggregates the information related to all available resources at one place. The Index Service can be built on top other Index Services and each child Index Service mapping specific type of resources. This concept of child Index Services leads to better management and organization of resources in the hierarchical manner. Each Grid Node i.e. GT4 WS-RF container has one Index Service supporting all the operations required by WS-ServiceGroup specification. By default the resources instantiated through resource homes are not registered with the Index Service. The resource home explicitly registers new creates resource instance with the Index Service; which aggregates all the resources at a single point. The registration process is done through ServiceGroupRegistrationClient. Below is the Java code snippet from the resource home to register the resource with the Index Service:

```
protected void add(ResourceKey key, Resource resource) {
super.add(key, resource);
ServiceGroupRegistrationClient regClient =
ServiceGroupRegistrationClient.getContainerClient();

ResourceContext ctx;
try {
   ctx = ResourceContext.getResourceContext();
} catch (Exception e) {
e.printStackTrace();
}
EndpointReferenceType epr=null;
try {
   URL baseURL = ServiceHost.getBaseURL();
   String instanceService = getInstanceServicePath();
   String instanceURI = baseURL.toString() + instanceService;
```

```
   epr = AddressingUtils.createEndpointReference(instanceURI, key);
} catch (Exception e) {
   e.printStackTrace();
}
// Location of the configuration file
String regPath = ContainerConfig.getGlobusLocation()
  + "/etc/org_sws_examples_services_buyer/registration.xml";

try {
   AddressBookResource buyerResource = (AddressBookResource) resource;
ServiceGroupRegistrationParameters params = ServiceGroupRegistrationClient
      .readParams(regPath);
params.setRegistrantEPR(epr);
Timer regTimer = regClient.register(params);
addressBookResource.setRegTimer(regTimer);
//regClient.register(params);
} catch (Exception e) {
   e.printStackTrace();
}
 }
```

Listing 9. Code snippet demonstrating registration of resource with Index Service

The resources have tight control the amount of information they want to advertise in the Index Service, how frequently the Index Service query their states to update itself and the polling interval. The Index Service is distributed registry; which keeps on updating itself regularly after specific intervals. Any change in the state of the resource or its deletion results in the notification of Index Service; which adjusts itself accordingly. This information is provided in the form configuration file '*registration.xml*' for each resource. Below is the sample configuration file.

```
<ServiceGroupRegistrationParameters
xmlns:sgc="http://mds.globus.org/servicegroup/client"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:agg="http://mds.globus.org/aggregator/types"
   xmlns="http://mds.globus.org/servicegroup/client" >

<RefreshIntervalSecs>30</RefreshIntervalSecs>
   <Content xsi:type="agg:AggregatorContent"
xmlns:agg="http://mds.globus.org/aggregator/types">
      <agg:AggregatorConfig xsi:type="agg:AggregatorConfig">
         <agg:GetMultipleResourcePropertiesPollType
            xmlns:bs="http://www...../AddressBookService" >
            <agg:PollIntervalMillis>20000</agg:PollIntervalMillis>
<agg:ResourcePropertyNames>….</agg:ResourcePropertyNames
<!—Resource Properties specific to the AddressBookService -->
  …….
</agg:GetMultipleResourcePropertiesPollType>
      </agg:AggregatorConfig>
    <agg:AggregatorData/>
   </Content>
</ServiceGroupRegistrationParameters>
```

Listing 10. XML code snippet of registration.xml

9. Web Services Developed

In the section 5.1 different components required in any Grid applications are discussed. In the agro-marketing use case along with discussed components, different stake holders are also involved such as buyer, seller, market etc. These components and stake holders are implemented

are either implemented as stateful Web services or simple stateless Web services based on its role in the overall functioning of the application. Below is the list of different Web services with their role and responsibilities, developed during the prototype.

1. **Seller Service:** The Seller Service implements the business logic necessary for the seller. This service expose different operations related to the seller resource such as instantiation of seller resources, monitoring and modification of different properties for seller resources. These operations are either executed manually by the seller or invoked automatically by other components during the course of the application as discussed in the different sequence diagrams.
2. **Buyer Service:** The Buyer Service is the implementation of our buyer stake holder. It instantiates the resource for a buyer, and provides different operation on the buyer resources. Instantiation of a buyer resource result in the registration of the Buyer with our system for future interactions. The service after receiving the preferences instantiates a new resource instance for the buyer. The service also registers these preferences with the DefaultIndexService of GT4 and provides operations to be invoked on this resource instance.
3. **Market Service:** This service acts as an intermediary between buyers and sellers. The role of the Market Service is more or less like a broker service among different stake holders. It queries the Grid Registry (i.e. Index Service provided by the Globus Toolkit) to obtain the list sellers and buyers trading in the same city/market and continuously monitors the trading preferences of buyers and sellers to perform trade. Market Service instantiate the new trade and transaction once it successfully matches the seller and buyer preferences. Initiation of the trade triggers sequence of events which results in the update of buyer and seller resources involved in the trade and notification to interested parties.
4. **GridManager Service:** The GridManager Service acts as a common factory service to instantiate different resources involved in the case study. The GridManager Service is implemented according to the Factory/Instance Collection Pattern and Master/Slave Pattern as discussed in the section 4.1. GridManager Service instantiate different resources through the specific service i.e. seller resource is instantiated through Seller Service. Interacting.
5. **VehicleFee Service:** VehicleFee Service is responsible for calculating any transportation charges incurred in case of indirect trade. The transportation charges are deducted from the final selling price and are paid by the seller.
6. **CropPrice Service:** CropPrice Service is used for retrieving the current market price of the crop involved in the trade. The CropPrice Service calculates the price of the crop based on different factors such as the type of the crop, season, its availability in the market, and its demand etc. This service is only invoked when the mode of trade is indirect.
7. **MarketingFee Service:** The MarketingFee Service is used for deducting any marketing fee incurred during the course of indirect trade. Similar to the transportation charges, marketing fee is deducted from the final selling price and is paid by the seller.

In the subsequent sections, details about various implemented services are given.

9.1 GridManagerService

The “GridManager” service as mentioned above is an interface provided to the Client application to instantiate resources related to different stateful Web services. The GridManager Service is implemented according to the Factory/Instance Collection Pattern and Master/Slave Pattern;

which instantiate appropriate resource according to the client preference. The intended service whose resource instance has to be created is specified by the client program.

9.1.1 WSDL Document for GridManagerService

It's important that the reader should have a fair understanding of the WSDL and how to write WSDL. The implementation of the any Web service must be according to the corresponding WSDL document as they both are part of contract to the user of the service. Below is an XML fragment from the WSDL document of the GridManager Web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GridManagerService"
targetNamespace=
"http://www.securingwebservices.org/namespaces/examples/GridManagerService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.securingwebservices.org/namespaces/examples/GridManagerSe
rvice"
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">
<types>
<xsd:schema
targetNamespace=http://www.securingwebservices.org/namespaces/examples/GridMana
gerService
xmlns:tns="http://www.securingwebservices.org/namespaces/examples/GridManagerSe
rvice"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import  namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
     schemaLocation="../../ws/addressing/WS-Addressing.xsd" />

  <!-- REQUESTS AND RESPONSES -->
  <xsd:element name="createResource">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="serviceName" type="xsd:string"/>
    </xsd:sequence>
  </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="GridManagerPortType">
  <operation name="createResource">
    <input message="tns:CreateResourceRequest"/>
    <output message="tns:CreateResourceResponse"/>
  </operation>
</portType>
```

```
</definitions>
```

Listing 11. WSDL file of GridManagerService

There are few important things to note about this WSDL file; which may differs from any normal WSDL doucument.

- This WSDL file is using a data type declared externally i.e. wsa:EndpointReference. The WSDL needs to import any externally declared data type used within the document, so that parsing tools can creates appropriate stubs and proxies. The imported data types are referenced with the fully qualified name; which requires declaring the appropriate namespace in the <wsdl:defination> element as an attribute.

```
<definitions name="GridManagerService"
     ……
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">

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

- The WSDL document for the GridManager Service exposes only one operation createResource which takes single String as a parameter wrapped in the data type createResource. Based on the value of the String parameter appropriate resource is instantiated and the corresponding EPR is returned to the client for future interaction.
- The GridManagerPortType is a collection of all the operations exposed by the GridManager Web service.

9.1.2 Implementation of GridManagerService

The WSDL document is a contract between the service and the client. Implementation of the service strictly follows this contract and it has one-to-one mapping of all functionality advertised through the WSDL. The implementation of the GridManager implements the only operation exposed by the WSDL document. The implementation can have any number of utility operations to support the mandatory operation. The mandatory method in the GridManager is the createResource() method, which instantiates the appropriate resource and returns the corresponding EPR. The main points related to the implementation are discussed below with reference to relevant implementation.

```
public CreateResourceResponse createResource(CreateResource request)
     throws RemoteException {
    // Retrive the service name from the client request
    String serviceName=request.getServiceName();

    BuyerResourceHome home = null;
    SellerResourceHome sellerHome = null;
    MarketResourceHome marketHome = null;
    ResourceKey key = null;
    String homeLoc=null;
    EndpointReferenceType instanceEPR;
      try {
      if(serviceName.equals("buyer")) {
        home = (BuyerResourceHome) getInstanceResourceHome(serviceName);
        key = home.create();
      }
//Similar kind of code for other services
catch (Exception e) {
        throw new RemoteException(e.getMessage(), e);
      }
      EndpointReferenceType epr = null;
      try {
        URL baseURL = ServiceHost.getBaseURL();
        String instanceService="";
if(serviceName.equals("buyer")) {
instanceService=(String)MessageContext.getCurrentContext().getService().getOption(
```

```
"buyerInstance");
}
// Similar kind of code for other services
      String instanceURI = baseURL.toString() + instanceService;
      epr = AddressingUtils.createEndpointReference(instanceURI, key);
    } catch (Exception e) {
      throw new RemoteException(e.getMessage(), e);
    }
  CreateResourceResponse response = new CreateResourceResponse();
  response.setEndpointReference(epr);
  instanceEPR=response.getEndpointReference();

  return response;
  }
```

Listing 12. createResource() method of GridManagerService

1. The createResource operation retrieves the name of the service from the client request for which the resource has to be instantiated. The serviceName is a private attribute of type String in the CreateResource class.
2. The getInstanceResourceHome is a private utility method to obtain the home of an appropriate resource. Each resource is instantiated through its corresponding home.
3. The object of ResourceHome is utilized to invoke create method of resource home object. The create() method creates a new resource instance and returns its unique identifier or the resource key. Notice that key is an object of the ResourceKey class. The properties related to resource key like the type of key, name of the key etc are defined in the JNDI file of the instance as discussed earlier.
4. Once the create method returns a key, then this key is used to construct the EPR of the WS-Resource. Remember

   EndPointReference=URL of the instance Service + Unique Resource Identifier.
5. The next step is to create the URL of the instance service. The URL of the instance service is created by adding the name of the instance service to the base URL of the container. The Factory service obtains the name of the appropriate instance service from the "wsdd" (Web service Deployment Descriptor) file.

```
String instanceURI = baseURL.toString() + instanceService;
```

   The "wsdd" file of GridManager service has an entry corresponding to the instanceService,

```
<parameter name="buyerInstance" value="sws/examples/BuyerService"/>
```

6. Once URL and resource key are obtained, the remaining step is to create the EPR to be return back to the client using the generated stub classes from the WSDL file.

```
epr = AddressingUtils.createEndpointReference(instanceURI, key);
CreateResourceResponse response = new CreateResourceResponse();
response.setEndpointReference(epr);
return response;
```

This utility method getResourceInstanceMethod() retrieves the information from the JNDI file associated to locate the appropriate resource home as discussed in the last section.

9.1.3 Resource Home and Resource

The GridManager service is stateless Web service which only instantiate different resources managed by other services. It doesn't manage any resource, thus there is no specific resource and resource home attached with the GridManager service.

9.2 BuyerService

Once the resource associated with the BuyerService is instantiated and the corresponding EPR is returned to the GridManagerClient. The GridManagerClient invokes different operations of the instance service referenced in the EPR; which may result in modification and update of the resource properties of the resource associated with the instance service.

9.2.1 WSDL Documents for the BuyerService

Various operations are exposed by the buyer service; which are invoked during the transaction such as setting the crop name, the crop quantity, crop variety, market etc. These entities related to the buyer are exposed as a Resource Properties of the Buyer resource and are declared in the Buyer.wsdl file.

```
<xsd:element name="BuyerCropName" type="xsd:string"/>
<xsd:element name="BuyerCropQuantity" type="xsd:float"/>
<xsd:element name="BuyerCropVariety" type="xsd:string"/>
<xsd:element name="BuyerMarket" type="xsd:string"/>
<xsd:element name="BuyerOfferedPrice" type="xsd:float"/>
<xsd:element name="BuyerAmountSpent" type="xsd:float"/>

<xsd:element name="BuyerResourceProperties">
<xsd:complexType>
   <xsd:sequence>
     <xsd:element ref="tns:BuyerCropName" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:BuyerCropQuantity" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:BuyerCropVariety" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:BuyerMarket" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:BuyerOfferedPrice" minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="tns:BuyerAmountSpent" minOccurs="1" maxOccurs="1"/>
   </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Listing 13. Resource Property declaration in Buyer.wsdl

The above mentioned code snippet shows how different resource properties of the buyer resource are declared in the WSDL document. The individual elements are declared as string, float etc. and then they are referenced in the complex element BuyerResourceProperty. The resource properties are declared according to the Document/literal style; which is mandatory according to the WS-Resource Framework specification version 1.0.

The BuyerResourceProperty element is coupled with the Web service through the “ResourceProperies” attribute of the <wsdl:portType> element. The “ResourceProperies” attribute of the <wsdl:portType> element is the extension of the WSDL 1.1 specification only meant for the WS-RF compliant containers and clients. The code snippet 18 demonstrates the association of a portType with a ResourceProperty.

```
<portType name="BuyerPortType" wsrp:ResourceProperties="tns:BuyerResourceProperties">
```

Listing 14. Association of portType with ResourceProperty

9.2.2 Implementation of BuyerResource

The resource declared in the WSDL document is implemented as a separate class. It is important, that the Factory service can instantiate the resource and the Instance service can locate and update the resource properties of the corresponding resource. The container locates any particular resource through its unique ID and its qualified name as declared in the WSDL document. The namespaces in the Web services are always error prone and these are normally declared in a separate interface for reusability and maintenance purposes.

```
public interface BuyerConstants {
```

```
public static final String NS =
"http://www...../namespaces/examples/BuyerService";
public static final QName RP_BUYERCROPNAME = new QName(NS, "BuyerCropName");
……
}
```

Listing 15. Code snippet of Java interface for namespace declaration

Resource properties and resources are referenced by their fully qualified name. The fully qualified name includes the namespace to which the resource property and resource belongs and its local name as shown in the above code snippet.

The namespace and local name of the resource and resource properties in the interface must be in accordance with the fully qualified name of the resource and resource properties in the WSDL document. It is possible to hard code these qualified names in the Web service and resource implementation class which can lead to unnecessary typo errors. However, use of separate interface is an elegant approach since this allows all declaration in a single place. In case of any change, only interface needs update, which considerably minimize the possibilities of errors.

Each resource is implemented as a separate class and the name of the resource class is declared in the JNDI configuration file. The resource class is very simple class only instantiating different private resource properties and providing corresponding accessor methods.

```
// BuyerResource class.
public class BuyerResource implements Resource, ResourceIdentifier,
    ResourceProperties, TopicListAccessor {
private String buyerCropName;
public Object initialize() {
this.key = uuidGen.nextUUID();
this.propSet = new SimpleResourcePropertySet(
      BuyerConstants.BUYERRESOURCE_PROPERTIES);
    try {
buyerCropNameRP=new SimpleResourceProperty(BuyerConstants.RP_BUYERCROPNAME);
setCropName("NULL");
buyerCropNameRP.add(buyerCropName);
this.propSet.add(buyerCropNameRP);
….
    } catch (Exception e) { …..}
        …..
  }
}
```

Listing 16. Code snippet from BuyerResource class

The code snippet above shows the declaration of a resource property set (i.e. the resource property set wraps all resource properties related to a single resource) and a particular resource property the “buyerCropNameRP”. The resource property set maps to the resource declaration in the WSDL document and thus shares the same fully qualified name.

The buyerCropNameRP is declared as an object of a SimpleResourceProperty type, provided by the GT4 implementation of the WS-RF. The SimpleResourceProperty eases the efforts to manage, monitor and update individual resource properties. The objects of SimpleResourceProperty can also be advertised as WS-Notification topics without extra efforts. The container monitors the states of these WS-Notification topics and generates notification messages on any state change. We will see WS-Notification in a subsequent section.

The SimpleResourceProperty has two constructors; one takes the qualified name of the resource property and other takes the object of ResourcePropertyMetaData. The ResourcePropertyMetaData is an interface and SimpleResourcePropertyMetaData is its concrete implementation. By default GT4 WS-RF implementation ignores the maximum and minimum occurrence of any element declared in the WSDL. The ResourcePropertyMetaData provides developer tight control on the cardinality of different resource properties within the resource. The

utility interface *BuyerConstants* explained earlier is utilized to initialize resource property set and different member resource properties.

As mentioned earlier the resource property set is the wrapper around different resource properties, use of GT4 provided API makes it easy to transform these objects in the corresponding XML document. This XML document is part of SOAP messages exchanged between a client and the service. In the implementation, different resource properties are initialized without any default values. It is possible to assign the default values directly to the resource properties but the more flexible approach is to declare private variables for each resource property and these variables are used to manage the state of resource properties. Each private variable has corresponding accessor methods which are indirectly used to modify and query values of resource properties at run time. For example, the resource property BuyerCropName declared in the WSDL has corresponding utility variable buyerCropName of type String.

```
private String buyerCropName;
public void setCropName(String cropName) {
  this.buyerCropName = cropName;
}
```

Listing 17. Accessor method for setting resource property

The buyerCropNameRP; which is the actual resource property is passed the utility variable buyerCropName. However, it is worth mentioning that the getters and setters are not strictly required when resource properties are instances of SimpleResourceProperty. We can directly assign the value to the resource property but this approach makes it difficult to manage the state of the resource property and the source of change. Below is the code assigning the value to the resource property buyerCropNameRP through the utility variable buyerCropName.

```
buyerCropNameRP.add(buyerCropName);
```

Listing 18. Assignment of Resource Property value using utility variable

Once the resource properties are initialized properly they are added to the resource property set. The same process has to be followed for initialization of all other resource properties, which collectively represent the state of the resource.

```
this.propSet.add(buyerCropNameRP);
```

Listing 19. Addition of resource property to resource property set

9.2.3 Implementation of BuyerResourceHome

The BuyerResourceHome is a typical resource home implemented on the lines discussed earlier and its implementation details are not repeated here. The main functionality of the BuyerResourceHome is outlined below.

- Creates a new instance of BuyerResource and passing the EPR to the GridManager
- Provides the querying and searching functionality of the resource based on the resource key
- Registers the newly created resources in the DefaultIndexService of Globus Toolkit 4.

9.2.4 Implementation of BuyerService

The BuyerService provides various operations to manage and alter the state of already created buyer resources through the EPR. The BuyerService is the interface between buyer resources and entities interested in its state i.e. client, GridManager Service, Market Service etc. The BuyerService is a stateful Web service exposing various business operations and few operations related to the management of the resources as specified in the WS-RF specification. The BuyerService does not implement WS-RF specific operations as they are provided by the WS-RF container; but it has to provide the signature of WS-RF specific methods supported by the service

in the WSDL file. Any service deployed in the WS-RF container is not stateful service unless it manages any resource and supports few of the WS-RF specific operations. The stateful service does not need to support all of the WS-RF specific operations and it can only supports subset of operations appropriate for its business logic. The implementation details of the BuyerService are pretty trivial, and most of the logical details are related to the deployment descriptor and the WSDL document. The "wsdd" file which is discussed earlier in the section 8.3 is specific to the Axis framework. The service informs the WS-RF container of its dependency on the WS-RF specific operations; which are implemented by the container through the parameter "providers". The parameter "providers" take space separated list of Java classes implementing various WS-RF specific operations. These classes are already registered with the WS-RF container and the container knows which class implements which operation. Below is the fragment from the "wsdd" highlighting the use of parameter "providers":

```
<parameter name="providers" value="GetRPProvider GetMRPProvider DestroyProvider"/>
```

The GT4 provides extension of the <wsdl:portType> in the form of attribute 'wsdlpp'. The 'wsdlpp' stands for 'WSDL pre processor'; which is precise way to add the details of the WS-RF specific operations in the WSDL supported by the service. Below is the portion of the WSDL document from the BuyerService showing the use of 'wsdlpp'.

```
<portType name="BuyerPortType"
  wsdlpp:extends="wsrpw:GetResourceProperty wsrpw:GetMultipleResourceProperties"
        wsrp:ResourceProperties="tns:BuyerResourceProperties">
```

Listing 20. Usage of WSDL pre processor (wsdlpp)

The developers should remember that this extension property is a utility feature only provided by GT4, for the convenience. In the final flattened file generated by GT4, the actual signature of the (i.e. messages and elements) related to wsrpw:GetResourceProperty will be inserted in the WSDL. It can be understood as a copy paste operation of picking elements from one WSDL file and placing them in another WSDL. The use of 'wsdlpp' restricts the reuse of the WSDL document across different implementations of WS-RF so it should be used with care. The actual signature of the 'wsrpw:GetResourceProperty' is shown below:

```
<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>
```

Listing 21. Signature of the wsrpw:GetResourceProperty

The GetResourceProperty and GetMultipleResourceProperties allow the clients to query and retrieve the current values of the resource properties of the resources, for details read section 8.3. These portTypes are also utilized by the Index Service to regularly query the resources for updated information regarding their state.

The business operations implemented by the BuyerService are pretty trivial. Each method looks up the resource referenced in the EPR through the resource home and then after locating the resource it invokes the appropriate action on it. Below is the code for updating the amount spent by the buyer in a single year; which can be used for statistical, accounting or taxation purposes (not implemented in the case study).

```
public AmountSpentResponse setAmountSpent(float amount) throws RemoteException
  {
    BuyerResource buyerResource = getResource();
```

```
    System.out.println("Old amount spent:"+ buyerResource.getAmountSpent());
    amount=amount+buyerResource.getAmountSpent();
    System.out.println("New amount spent:"+ amount);
    Float amountSpeant=new Float(amount);
    buyerResource.setAmountSpent(amountSpeant);
    return new AmountSpentResponse();
  }
```

Listing 22. BuyerService operation setAmountSpent()

The reader can now appreciate the use of accessor methods for the utility private variables corresponding to each resource property, which were implemented in the BuyerResource class. These utility methods provide more elegant solution to update the resource properties. These utility methods can validate the values before assigning to the resource properties. This validation can be of various nature such as the appropriate range for numerical values, decimal rounding for floating points, regular expressions for the String values. Below is the implementation of the method from the BuyerService; which is initializing various properties related to the buyer resource via utility accessor methods.

```
public BuyerPropertiesResponse setBuyerProperties(BuyerProperties params)
throws RemoteException {
String cropName=params.getCropName();
float cropQuantity= params.getCropQuantity();
String cropVariety=params.getCropVariety();
String market=params.getMarket();
float offeredPrice=params.getOfferedPrice();
float amountSpent=params.getAmountSpent();
Float cQ=new Float(cropQuantity);
Float oP=new Float(offeredPrice);
Float aS=new Float(amountSpent);
BuyerResource buyerResource = getResource();
buyerResource.setCropNameRP(cropName); buyerResource.setCropQuantityRP(cQ);
buyerResource.setCropVarietyRP(cropVariety);
buyerResource.setMarketRP(market);
buyerResource.setOfferedPriceRP(oP);
buyerResource.setAmountSpentRP(aS);
return new BuyerPropertiesResponse();
}
```

Listing 23. Operation setBuyerProperties() to update Buyer resource

BuyerProperties is a complex data type which is declared in the WSDL document of the BuyerService. A wrapper Java class is generated for this data type when the tool “wsdl2Java” is used. The object of this class is used within the actual implementation of the case study and by the client application rather than working with raw XML documents through DOM or SAX API.

9.3 SellerService

The SellerService manages and monitors the resources related to the seller and its implementation is very similar to the BuyerService. Hence, the explanation of the SellerService is not repeated here. The complete WSDL document and the implementation of the SellerService is included in the downloadable source code.

9.4 MarketService

The MarketService is an intermediate between the ByerService and the SellerService. The MarketService provides the required platform to carry out any trade. The MarketService orchestrate different services in the pre-set manners for successful transactions, updating the involved resources, notifying the interested parties.

9.4.1 WSDL Document for the MarketService

The WSDL document of the MarketService is similar to any other WSDL document, with few business operations, subset of WS-RF specific operations and single resource. The WSDL document for the MarketService needs no further explanation and is not repeated here. The resource, resource home and utility interface are implemented according to the Implied Resource Pattern as discussed in the section 4.1.

9.4.2 Implementation of MarketResource and MarketResourceHome

The implementation of the MarketResourceHome has exactly the same business logic and nearly the same operations as discussed in the BuyerService section 9.2. The MarketResource is instantiating the resource property set and different resource property elements on the same line as discussed in the BuyerResource. The implementation of these classes are not repeated here, but the source code is available for download.

9.4.3 Implementation of MarketService

The MarketService implements few business operations to manage resource but the most important methods in the MarketService are the fireXpathQuery() and storeEntries() method. In this section only these two methods are discussed. Familarity of these methods will help to understand the working of IndexService i.e. Grid Registry provided by the GT4.

9.4.3.1 The fireXpathQuery() Method

The fireXpathQuery() method is used to construct and fire appropriate XPath query, to query the IndexService (i.e. Grid Registry). The main purpose of this operation is to match trading preferences of buyers and sellers registered in the same market.

```
QueryResourcePropertiesResponse fireXpathQuery(String type,String marketCity) {
String indexURI="http://10.100.64.64:8080/wsrf/services/DefaultIndexService";
String xpathSellerQuery,xpathBuyerQuery="";
EndpointReferenceType indexEPR = new EndpointReferenceType();
try {
    indexEPR.setAddress(new Address(indexURI));
} catch (Exception e) {
e.printStackTrace();
}
// Get QueryResourceProperties portType
WSResourcePropertiesServiceAddressingLocator queryLocator;
    queryLocator = new WSResourcePropertiesServiceAddressingLocator();
    QueryResourceProperties_PortType query = null;
    try {
      query= = queryLocator.getQueryResourcePropertiesPort(indexEPR);
    } catch (Exception e) {
      e.printStackTrace();
    }
if(type.equals("seller")) {
xpathSellerQuery ="//*[local-name()='Entry'][./*/*/*[local-
name()='SellerMarket']/text()='"+ marketCity +"']";
} else {
xpathSellerQuery ="//*[local-name()='Entry'][./*/*/*[local-
name()='BuyerMarket']/text()='"+ marketCity +"']";
}
    // Create request to QueryResourceProperties
    QueryExpressionType queryExpr = new QueryExpressionType();
    try {
      queryExpr.setDialect(new URI(WSRFConstants.XPATH_1_DIALECT));
    } catch (Exception e) {
      e.printStackTrace();
    }
    queryExpr.setValue(xpathSellerQuery);
    QueryResourceProperties_Element queryRequest = new QueryResourceProperties_Element(
        queryExpr);
```

```
    // Invoke QueryResourceProperties
    QueryResourcePropertiesResponse queryResponse = null;
    try {
      queryResponse = query.queryResourceProperties(queryRequest);
    } catch (RemoteException e) {
    e.printStackTrace();
    }
    return queryResponse;
  }
```

Listing 24. Operation fireXpathQuery() for querying Index Service

The fireXpathQuery() operation is bit complicated method and good understanding of its implementation is important for the understanding of the role played by the MarketService. Below are the main points related to the fireXpathQuery() operation.

1. Firstly the fireXpathQuery() operation retrieves the URL of the IndexService. This URL is used to create the EPR of the IndexService. This new created EPR is used to invoke different operations on the IndexService.
2. The fireXpathQuery() invokes different operations on the IndexService through Axis generated utility classes, i.e. WSResourcePropertiesServiceAddressingLocator and QueryResourceProperties_PortType, which serialize and de-serialize SOAP request and response messages before transforming down the wire.
3. The next step is to create an appropriate XPath query to be fired. However, the version of XPath query being used is to be specified as there are two different versions of XPath. The XPath version used for the query should match the XPath supported by the WS-ServiceGroup. This purpose is achieved by using the dialect.
4. The XPath query is used to create an appropriate request message. This newly created request message is passed to the queryResourceProperties(..) operation of the QueryResourceProperties_PortType class. The QueryResourceProperties_PortType the local stub for the remote Web service.
5. The queryResourceProperties(..) returns the result of the query wrapped in the QueryResourcePropertiesResponse object.

9.4.3.2 The storeEntries() Method

After receiving the response from the queryResourceProperties(..) in the fireXpathQuery, the next step is to get the entries matching the query crieria. Remember that the DefaultIndexService of GT4, is actually an Aggregrator Service i.e. its function is to aggregrate several resource properties together according to the WS-ServiceGrooup specifications. All entries are retrieved from the response by calling queryResponse.get_any() method; which returns an array of MessageElement type.

```
MessageElement[] buyerEntries = queryResponse.get_any();
```

The MessageElement is the Axis framework specific Java mapping of data type xsd:any. The data type xsd:any is used to pass raw XML between services particularly when the contents of the XML are dynamic and changes at run time. Hence, we need to de-serialize the MessageElement entries to invoke different operations on them and particularly to retrieve the EPR of the resources for further point-to-point communication. The business logic of retrieving the information from individual entries is implemented in the storeEntries(..) operation. Below is the implementation of the storeEntries(..) operation.

```
void storeEntries(QueryResourcePropertiesResponse queryResponse,String type,
MessageElement[] entries) {
  Vector tmpVector=new Vector();
  EndpointReferenceType[] tempEPR=new EndpointReferenceType[entries.length];
```

```
for(int m=0;m<entries.length;m=m+1) {
    tempEPR[m]=new EndpointReferenceType();
  }
  for (int i = 0; i < entries.length; i++)   {
  try {
    int j=0;
    EntryType entry = (EntryType) ObjectDeserializer.toObject(
    entries[i], EntryType.class);
    if (type.equals("seller")) {
      tempEPR[i]=entry.getMemberServiceEPR();
    } else {
      tempEPR[i]=entry.getMemberServiceEPR();
    }
    AggregatorContent content = (AggregatorContent) entry.getContent();
    AggregatorData data = content.getAggregatorData();
    String cropName = data.get_any()[0].getValue();
    tmpVector.addElement(cropName);
    ….
  } catch (Exception e) {…}
….
}
```

Listing 25. Operation storeEntries()for de-serialization of MessageElement entries

First an array of the MessageElements is de-serialized into an object of EntryType class. From the object of EntryType different operations are invoked to retrieve information related to that the particular entry i.e. EPR. AggregatorContent class is used to retrieve the aggregated data within a particular "entry" object of EntryType class. Since, the actual data i.e. resource properties are required for match making; which is the key to initiate the trade.

```
AggregatorContent content = (AggregatorContent) entry.getContent();
AggregatorData data = content.getAggregatorData();
```

The "data" object obtained, is actually an array of the information advertised by the resource within the Index Service in the form of String. Hence, each entry in the array is reference to the specific resource property. The registration.xml file is used to set the resource properties to be advertised in the IndexService; and is discussed in the section 8.6.

String cropName = data.get_any()[0].getValue(); is utilized

The data.get_any()[0].getValue() retrieves the very first entry of the array which has been returned, which is of String type. Similarly, the process has to be repeated to obtain the values for other entries. Since the position of the element can change; which requires the appropriate change in the 'index' passed to the get_any() operation. Once all entries are retrieved they are stored in a Vector, whose elements are accessed later to mach if the requirements of any buyer are satisfied by any seller. Operation performTrade() uses this Vector in which the buyer and seller entries were stored and then performs the trade contingent on the following conditions being met

- Crop required by any buyer is being offered by any seller
- Quantity is appropriate
- Crop Variety is matching
- Offered price of buyer is higher or equal to expected price of seller.

If these conditions are met, then the performDirectTrade0 function is invoked by passing the seller & buyer EPR, the quantity to be purchased and the amount, which will be offered to seller. Operation performDirectTrade(), is a normal function which utilizes the EPR's of the buyer and the seller to adjust the quantity and the amount spent by buyer and amount earned by the seller. Thus, by adjusting these resource properties simulate the successful trade and transaction.

For example if a seller has 60Kg of rice offered at 10Rs/kg and its preference match that of a buyer which needs this 60Kg at 12Rs/Kg, the SellerCropQuantity ResourceProperty for this instance will be set to 0 and BuyerCropQuantity will be also set to 0, since the seller has sold as much as it wanted to and buyer has purchased as much as it wanted to. However, the

SellerEarning ResourceProperty will be set to Rs 60 X 12= Rs 720 and the BuyerAmountSpent ResourceProperty will be set to Rs 60 X 12 = Rs 720.

9.5 Services for Indirect Trading

The market service also takes care of performing trading for sellers interested in indirect trading. This is achieved with the help of simple services by invoking operations on them by passing suitable parameters. These services manipulate a resource properties associated with the EPR of the seller for whom the trade is being performed. Hence, they have functions which are invoked by passing EPR of the seller. Crop quanity, crop name, crop variety etc are also passed depending upon the provided functionality.

9.5.1 CropPriceService

CropPriceService is a simple Web service, which returns the price to be offered to any farmer which is interested in indirect mode of trade. For any crop name which is provided as input, it provides the price per kg which is offered. It does not have a resource factory with it, as the function is invoked by passing input parameters. It is based on singleton resource instance pattern. This multiplied by quantiy returns the total amount for this crop variety. The service further invokes the setEarning() method of SellerService to set the earning for this crop. The code of the service is easy to understand and comprises of just one java file namely CropPriceService.java along with other additional files related to container deployment.

9.5.2 MarketFeeService

MarketFeeService is based on similar pattern as CropPriceService. It deducts the marketing fee, which is imposed on the earning of the seller. As explained above, this also takes seller EPR as input. It utilizes this EPR to obtain the earning of the seller and deducts the market fee as applicable and invokes operation setEarning() of SellerService to set the earning for the seller after deduction of market fee.

9.5.3 VehicleFeeService

VehicleFeeService imposes vehicle fee levied on the seller for using vehicle inside market area for transport of crop. It takes seller EPR and crop quantity transported as input and then invokes setEarning() operation to set the earning for this seller after deducting vehicle fee. The service as mentioned.

9.6 Client

The end user interacts with the developed Web services with the help of a Java based Client program.The end user executes this client program by providing certain command line arguments. These command line arguments are different for a seller,buyer and market. Hence, from the common code related to intializing some paramters, the discussion for the client will be dealt in 3 different sections with perspective for seller, buyer and market.

9.6.1 SellerPerspective

(a) An end user interested in executing the client as seller has to provide following command line arguments:

1. URL of GridManager Service
2. Trader type namely seller in this case.
3. Crop name offered for sale
4. Crop variety

5. Crop quantity
6. Expected price
7. Trade mode like if seller wants to do direct trading with a buyer or wants to offer crop to amrket service for sale
8. Preferred trading location.

The command line arguments are assigned to different variables like

```
cropName=args[2];
cropQuantity=args[3];
cropVariety=args[4];
//Unrelated code in between
if(serviceType.equals("seller"))
{
  market=args[5];
  expectedPrice=args[6];
  tradeMode=args[7];
  earning="0";
  f2=new Float(expectedPrice);
}
```

Listing 26. Code snippet demonstrating assignment of command line arguments

(b) Using values assigned to these variables, an object of the SellerProperties is created.

```
sp=new SellerProperties(cropName,cq,cropVariety,op,market,tradeMode);
```

(c) Next a reference to the GridMangerPortType is obtained to create a new instance of the Seller Resource. This is achieved by using the EPR of the GridManager service, which can be created by using the URI of it.

```
GridManagerPortType gridManager;
SellerPortType seller;
gridManagerEPR = new EndpointReferenceType();
gridManagerEPR.setAddress(new Address(gridManagerURI));
gridManager = gridManagerLocator.getGridManagerPortTypePort(gridManagerEPR);
```

Listing 27. EPR retrieval of GridManagerService

(d) Security related parameters are to be initialized in order to encrypt the exchange of SOAP message between the Client and the GridManagerService. Here, Message Level Security is being utilized. Hence, only the body of the message is encrypted leaving the header untouched.

```
((Stub)gridManager)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
((Stub) gridManager)._setProperty(Constants.AUTHORIZATION,
NoAuthorization.getInstance());
```

Listing 28. Initialization of security related parameters for SOAP message encryption

(e)The GridManagerService acts as a common factory for the Seller, Buyer and Market Service. Its create method will be invoked by passing an object of the stub class CreateResource, which was discussed in the section related to GridManagerService. The constructor of CreateResource class expects a String argument namely serviceType, to identify which stake holders Resource instance has to be created and initialized. In this case the serviceType argument will have value as “seller” to indicate that a new Seller Resource has to be created.

```
try
{
createResponse = gridManager.createResource(new CreateResource(serviceType));
}
```

The EPR of a Resource instance is to be returned after creating it. The EPR returned will be stored in an object of createResponse. The client needs to inform the GridManagerService about the stakeholder of a newly created resource instance namely the seller, buyer or the market.

(f) Once the EPR is returned and stored in the object createResponse, operations can be invoked on this Seller Resource to modify the default values and set the trading parameters as indicated by

the command line arguments provided by the end user. This is done, by first obtaining a reference to the SellerService port type, setting of security related parameters to secure the conversation, and then invoking operations exposed by the SellerService namely setSellerProperties() by passing the object of SellerProperties class created and initialized above.

```
if(serviceType.equals("seller"))
{
  sellerInstanceEPR= createResponse.getEndpointReference();
  notificationEPR=sellerInstanceEPR;
seller = sellerInstanceLocator.getSellerPortTypePort(sellerInstanceEPR);
((Stub)seller)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
((Stub) seller)._setProperty(Constants.AUTHORIZATION, NoAuthorization.getInstance());
try
{
seller.setSellerProperties(sp);
}
catch(Exception e)
{
  e.printStackTrace();
}
       }
```

Listing 29. Invocation of operation setSellerProperties of SellerService

(g) The EPR of the Seller Resource is created and written in a file, will be utilized by a client responsible for providing notification related information when changes take place in the value of SellerResourceProperties; i.e the sellerStatus. After writing EPR in a file, the client NotificationListenerClient responsible for listening to notifications is invoked by invoking its operation startSubscription by passing serviceType and filename which contains the EPR of this newly created Resource.

```
if(!serviceType.equals("market"))
{
  Random rand=new Random();
  int i=rand.nextInt();
  Integer number=new Integer(i);
  String fileName=number.toString()+".epr";
  Client cl=new Client();
  cl.writeEPR(notificationEPR,fileName);
  NotificationListenerClient notificationListener = new     NotificationListenerClient();

  notificationListener.startSubscription(serviceType,fileName);
}
}
catch (Exception e)
{
  e.printStackTrace();
}
}
void writeEPR(EndpointReferenceType epr,String fileName) throws
IOException,SerializationException {

    // invent a filename
    //String filename = "note-" + 785 + ".epr";
    FileWriter fw = null;
    try
    {
     fw = new FileWriter(fileName);
      // write the EPR into the file
   ObjectSerializer.serialize(fw, epr, RESOURCE_ENDPOINT);
        fw.write('\n');
    }
    finally
    {
      if(fw != null)
      {
        try
  {
         fw.close();
```

```
        }
        catch (Exception e) {}
      }
    }
  }
```

Listing 30. Code snippet for writing EPR in file

9.6.2 BuyerPerspective

(a) An end user interested in executing the client as buyer has to provide following command line arguments:

1. URL of GridManager Service
2. Trader type namely buyer in this case
3. Crop name interested in purchasing
4. Crop variety
5. Crop quantity
6. Offered price
7. Preferred trading location

The command line arguments are assigned to variables like we have assigned in case of Seller perspective. Few other variables will be also assigned, e.g.

```
if(serviceType.equals("buyer"))
{
   amountSpent="0";
   market=args[5];
   offeredPrice=args[6];
   f2=new Float(offeredPrice);
}
```

Listing 31. Code snippet demonstrating assignment of command line arguments

(b) Using values assigned to these variables, an object of the BuyerProperties is created.

```
buyerProperties=new BuyerProperties(cropName,cq,cropVariety,market,op);
```

(c) Next a reference to the GridMangerPortType is obtained to create a new instance of the Buyer Resource. The explanation and the steps are as similar as we have seen for Seller Perspective till step (e), where the serviceType will be ‘buyer” instead of “seller”.

(d) The EPR, which will be returned, will be of Buyer Resource, hence operations exposed by BuyerService, namely setBuyerProperties() will be invoked to replace the default initialized value of the Buyer Resource with the arguments specified by the end user. After this the explanation related to writing EPR in a file and notification is same as that of SellerResource.

```
try
{
buyer.setBuyerProperties(buyerProperties);
}
catch(Exception e)
{
  e.printStackTrace();
}
```

Listing 32. Invocation of operation setBuyerProperties of BuyerService

9.6.3 Market Perspective

(a) An end user interested in executing the client as market has to provide following command line arguments:

1. URL of GridManager Service
2. Trading location for which it wishes to trade.

(b) Most of the explanation related to execution of a client from Market perspective remains same as Seller Perspective and Buyer Perspective. The only noticible difference being that after obtaining the EPR of the Market Resource, the operation invoked will be setMarketProperties(marketProperties); marketPortType.setMarketProperties(marketProperties);

Once the property is set, then trading will begin for the specified trading location. Sellers and buyers registered with the DefaultIndexService will be matched for similar trading preferences and trade will be done. In case of indirect trade mode for some sellers, suitable amount will be offered and its Earning Resource Property will be modified to reflect its earning. WS-RF offers functionality related to Notifications using WS-Notification, whereby a client is delivered messages whenever there is a change in the Resource Property to which it subscribes.

9.7 Notification Listener Client

A client to subscribe and receive notification messages is developed.

```
package org.sws.examples.clients.GridManager;

public class NotificationListenerClient
    extends BaseClient
    implements NotifyCallback {

static String SellerNS =
"http://www.securingwebservices.org/namespaces/examples/SellerService";
static QName SellerRESOURCE_ENDPOINT = new QName(SellerNS,"SellerServiceEndpoint");

  static String BuyerNS =
"http://www.securingwebservices.org/namespaces/examples/BuyerService";
static QName BuyerRESOURCE_ENDPOINT = new QName(BuyerNS,"BuyerServiceEndpoint");

NotificationProducer producerPort=null;

final static WSResourcePropertiesServiceAddressingLocator locator =
      new WSResourcePropertiesServiceAddressingLocator();
public void startSubscription(String serviceType,String fileName) {

   String endpointString="";
    try {
      NotificationConsumerManager consumer = null;
      consumer = NotificationConsumerManager.getInstance();
      consumer.startListening();
      EndpointReferenceType consumerEPR =consumer.createNotificationConsumer(this);
      Subscribe statusRequest = new Subscribe();
      statusRequest.setUseNotify(Boolean.TRUE);
      statusRequest.setConsumerReference(consumerEPR);
      TopicExpressionType statusExpression = new TopicExpressionType();
      statusExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
      if(serviceType.equals("seller"))
      {
       statusExpression.setValue(SellerConstants.RP_SELLERSTATUS);
      }
      else
      {
       statusExpression.setValue(BuyerConstants.RP_BUYERSTATUS);
      }
      statusRequest.setTopicExpression(statusExpression);
      WSBaseNotificationServiceAddressingLocator notifLocator =
          new WSBaseNotificationServiceAddressingLocator();
         try
   {
     FileInputStream fis = new FileInputStream(fileName);
     File file=new File(fileName);
     file.deleteOnExit();
     endpoint=null;
     endpoint = (EndpointReferenceType) ObjectDeserializer.deserialize(new
InputSource(fis),
     EndpointReferenceType.class);
```

```
      if(serviceType.equals("seller"))
      {
        endpointString = ObjectSerializer.toString(endpoint,SellerRESOURCE_ENDPOINT);
      }
      else
      {
        endpointString = ObjectSerializer.toString(endpoint,BuyerRESOURCE_ENDPOINT);
      }
      System.out.println("\nEND POINT STRING:"+ endpointString);
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
producerPort = notifLocator.getNotificationProducerPort(endpoint);
EndpointReferenceType statusSubscriptionEPR =
producerPort.subscribe(statusRequest).getSubscriptionReference();
System.out.println("Waiting for notification. Ctrl-C to end.");
    }
    catch (Exception e) {
  e.printStackTrace();
    }
    while (true) {
      System.out.println("Waiting for notification. Ctrl-C to end.");
      try {
        Thread.sleep(30000);
      }
      catch (Exception e) {
      e.printStackTrace();
        //System.out.println("Interrupted while sleeping.");
      }
    }
  }
  public void deliver(List topicPath,
                      EndpointReferenceType producer,
                      Object message) {
    ResourcePropertyValueChangeNotificationType changeMessage =
        ( (ResourcePropertyValueChangeNotificationElementType) message).
        getResourcePropertyValueChangeNotification();
  if (changeMessage != null) {
  String msg=changeMessage.getNewValue().get_any()[0].getValue();
if(msg.equals("Resource Instance will be deleted since your trading has been
completed")||
msg.equals("\nYour purchasing is over!Your resource instance will be
destroyed")||msg.equals("\nYour crop is sold!Your resource instance will be destroyed"))
  {
    System.exit(0);
  }
      System.out.println("\nA new notification has arrived");
      System.out.println(changeMessage.getNewValue().get_any()[0].getValue());
    }
  }
}
```

Listing 33. Java code of NotificationListenerClient

1. The client is a notification consumer. It will receive notifications, which will be sent by the services, BuyerService and SellerService. The services acting as notification producer will invoke Notify operation on this client. The client runs in a daemon mode to receive the notifications at any time. Thus it acts like a server on which operations can be invoked any time. In order to attain this property NotificationConsumerManager class is utilized.

   ```
   NotificationConsumerManager consumer = null;
   consumer = NotificationConsumerManager.getInstance();
   consumer.startListening();
   ```

2. The client starts listening to the incoming notifications. In order to identify the end point at which the notification has to be delivered by the Seller or Buyer service, a proper address has to be assigned to this client. This is achieved by creating an EPR of the client.

```
EndpointReferenceType consumerEPR=consumer.createNotificationConsumer(this);
```

3. Next we send a request to the services to start sending the notification by using the standard Notify operation. The consumer EPR is passed to identify the end point at which the notification is to be delivered.

```
  Subscribe statusRequest = new Subscribe();
  statusRequest.setUseNotify(Boolean.TRUE);
  statusRequest.setConsumerReference(consumerEPR);
```

4. The client needs to inform the Topic to which it wishes to subscribe. In this case the client subscribes to different resource properties depending upon the service to which it subscribes. As for example, if the client program is executed as a seller, then the Resource Properties will be SellerStatus.

```
TopicExpressionType statusExpression = new TopicExpressionType();
  statusExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
 if(serviceType.equals("seller"))
  {
statusExpression.setValue(SellerConstants.RP_SELLERSTATUS);
  }
  else
  {
statusExpression.setValue(BuyerConstants.RP_BUYERSTATUS);
  }
```

5. A reference to the notification producer is obtained for sending subscription request by obtaining reference to the NotificationProducerportType, implemented by the remote service like the BuyerService or SellerService. This is done by using extends tag in the WSDL, where the portType of these service extend the NotificationProducerPortType. The notification to be delievered is for a service based on factory pattern. Thus, EPR of the resource instance to be subscribed is to be obtained. This is achieved by reading from a file, which is created by the Client.java program. After reading EPR subscription a request is sent to the remote service.

```
WSBaseNotificationServiceAddressingLocator notifLocator =
          new WSBaseNotificationServiceAddressingLocator();
        try
   {
     FileInputStream fis = new FileInputStream(fileName);
     File file=new File(fileName);
     file.deleteOnExit();
     endpoint=null;
endpoint = (EndpointReferenceType) ObjectDeserializer.deserialize(new InputSource(fis),
             EndpointReferenceType.class);
     if(serviceType.equals("seller"))
     {
       endpointString = ObjectSerializer.toString(endpoint,SellerRESOURCE_ENDPOINT);
     }
     else
     {
       endpointString = ObjectSerializer.toString(endpoint,BuyerRESOURCE_ENDPOINT);
     }
     System.out.println("\nEND POINT STRING:"+ endpointString);
   }
   catch(Exception e)
   {
     e.printStackTrace();
   }
      producerPort = notifLocator.getNotificationProducerPort(endpoint);
```

6. For every subscription request, a unique EPR is created. This EPR can be used to subscribe, pause and terminate subscription. This is achieved by

```
EndpointReferenceType statusSubscriptionEPR =
producerPort.subscribe(statusRequest).getSubscriptionReference();
```

7. The client starts listening to subscriptions till the time key is pressed. Incoming notifications are handled by a different method called deliver. The deliver method uses three parameters

namely the topicPath used for creating subscription, the EPR of a producer of notification and message which is sent by the service.

Our Resource classes used ResourcePropertyTopic for notification in ResourceProperties. Hence, the notification messages are of ResourcePropertyValueChangeNotificationType. This further embeds message of ResourcePropertyValueChangeNotificationType in itself. The object of this class contains the new resource property value.

10. Execution scenario

In this section the trading process is displayed by execution of various services depending on the preferences indicated by a client. Data shown in various tables is taken as input to execute various services part of Grid business process. Following table indicates preferences of the buyers:

Buyer	CropName	CropVareity	Quantity	PriceOffered Rs/kg	Trading Location
1	rice	basmati	50 Kg	100	Mumbai
2	wheat	sindhi	20 Kg	80	Delhi
3	rice	carolina	100 Kg	90	Mumbai
4	wheat	kamla	10 Kg	70	Bhopal
5	potato	katahdin	60 kg	60	Kolkatta

Table 1. Data about buyers

The data corresponding to various sellers is shown in the table as under:

Seller	CropName	CropVariety	Quantity	Price Expected Rs/Kg	Trading Location	Trade Mode
1	rice	arboria	100 Kg	65	Mumbai	Indirect
2	wheat	kamla	50 Kg	60	Bhopal	Direct
3	wheat	dandi	70 Kg	50	Mumbai	Direct
4	soyabean	monetta	40 Kg	70	Delhi	Indirect
5	corn	indian	110 Kg	55	Amritsar	Direct
6	potato	katahdin	60 Kg	10	Kolkatta	Indirect
7	rice	basmati	120 Kg	120	Mumbai	Direct
8	onion	nasikred	50 Kg	8	Mumbai	Direct

Table 2. Data about sellers

In the following table, locations of various markets are shown:

Location
Mumbai
Delhi
Bhopal
Calcutta
Amritsar

Table 1. Data about locations of various markets

10.1 Execution of Seller Grid Service

Now when the GridManagerClient is executed for each of the above given entities to perform trade, following situation will emerge:

- By looking at the data above it can be inferred that for direct trade, seller and buyers interested in trading in Bhopal will trade with each other. Similarly, trading will take place for sellers interested in indirect trading irrespective of location of their trading.
- A seller expresses his intention to offer his crop for sale by running the client program, Client.java by providing command line arguments.These arguments are in the following order:
  - URL of the GridManager service
  - seller: This word is provided to indicate client wants to have an instance of a seller service
  - Crop Name: The crop seller wishes to sell
  - Crop Quantity: The quantity of a crop, which seller is interested in selling
  - Crop Variety: The variety of a crop
  - Trade Location: The location where the seller wishes to trade
  - Expected Price/Kg: The amount which a seller wishes to obtain per kg of a crop
  - Trade Mode: If seller directly wants to sell directly to a buyer then direct mode is adopted, else indirect mode is adopted.

In the following example, an intention of a seller is shown, who is ready to sell 50 kg. of wheat at Bhopal location, and expecting minimum of Rs. 60 per kg.

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager seller wheat 50 kamla
Bhopal 60 direct
```

Execution of seller grid service will start and will wait for appropriate offer to come from a potential buyer. Seller grid service will receive notification, once an appropriate offer is floated by a buyer. Initially, following output will be generated:

```
END POINT STRING:<ns1:SellerServiceEndpoint xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.securingwebservices.org/namespaces/examples/SellerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/sws/examples/SellerSe
rvice</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:SellerResourceKey>a1662350-1628-11db-9878-ce21816644d9</ns1:SellerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:SellerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

10.2 Execution of Buyer Grid Service

Similarly a buyer can express his intention to purchase crop by running Client.java but with a different set of command line arguments:

- URL of the GridManager service
- buyer: This word is provided to indicate that a client would like to have an instance of a buyer service
- Crop Name: The crop buyer wishes to buy
- Crop Quantity: The crop quantity buyer is interested in purchasing
- Crop Variety: The crop variety of the crop
- Trade Location: The location where the buyer wishes to trade
- Offered Price/Kg: The amount which a buyer would like to obtain per kg of a crop.

As shown in the following screen, one buyer is ready to purchase 10 kg of wheat at Rs. 70 per kg. Thus, buyer grid service will also wait for appropriate match to happen in the market located at Bhopal.

```
$java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager buyer wheat 10 kamla
Bhopal 70
```

Here, execution of buyer grid service will start and it will wait for appropriate match to be identified in a market located at Bhopal.

```
END POINT STRING:<ns1:BuyerServiceEndpoint xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.securingwebservices.org/namespaces/examples/BuyerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/sws/examples/BuyerSer
vice</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:BuyerResourceKey>3136b260-1629-11db-9878-ce21816644d9</ns1:BuyerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:BuyerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

10.3 Execution of Market Grid Service

Whenever the buyers and sellers express their intentions to trade, their resource instances are also registered with the DefaultIndexService. When MarketService start its operations, it will query DefaultIndexService to obtain the buyers and sellers interested in trading. The Client.java program is executed with a different set of command line arguments.

- URL of the GridManager service.
- market- This word is provided to indicate that the client would like to have an instance of a market service.
- Trade location- The location of a market where the actual trade will occur.

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager market Bhopal
$
```

Once market service of a particular location starts executing, it tries to match potential buyers and sellers. If an appropriate buyer is found for a seller, approproate notification is sent to both of them to inform them about success of entering into a trade deal. Otherwise both of them will wait for a potential purchaser of a crop. In case of indirect trade mode, the seller receives various types of notifications, which are computed for this trade.

10.4 Notification to Seller Service

After initiation of a market service for Bhopal, matching will performed for buyer(s) and seller(s) having preference for Bhopal. Based on the match found for a seller and a buyer for wheat, following notification will be generated for a seller service:

```
A new notification has arrived

Your crop sold till now is 10.0Kg

A new notification has arrived

Your crop has been sold recently

A new notification has arrived

Your crop quantity still remaining to be sold is:40.0Kg
```

```
A new notification has arrived

Amount Offered to you per Kg is Rs70.0

A new notification has arrived

Crop Quantity Which will be purchased by a buyer is 10.0Kg

A new notification has arrived

Your Earning is Rs700.0
Waiting for notification. Ctrl-C to end.
```

10.5 Notification to a buyer service

Following notification will be generated for a buyer service:

```
A new notification has arrived

The Crop Quantity which you will purchase from a seller is 10.0Kg

A new notification has arrived

Amount Offered by you per Kg is:Rs70.0

A new notification has arrived

Amount spent by you in current purchase is Rs700.0
```

In the following example, one seller is interested in selling potato of katahdin variety in Kolkatta. The interactions among various services is shown in subsequent snap shots:

10.6 Execution of Seller Service

```
$java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager seller potato 60 katahdin
Kolkatta 10 indirect


END POINT STRING:<ns1:SellerServiceEndpoint xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.securingwebservices.org/namespaces/examples/SellerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/sws/examples/SellerSe
rvice</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
  <ns1:SellerResourceKey>faea5e70-162b-11db-9878-ce21816644d9</ns1:SellerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:SellerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

10.7 Execution of Buyer Service:

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager buyer potato 60 katahdin
Kolkatta 60


END POINT STRING:<ns1:BuyerServiceEndpoint xsi:type="ns2:EndpointReferenceType"
xmlns:ns1="http://www.securingwebservices.org/namespaces/examples/BuyerService"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/03/addressing">
 <ns2:Address
xsi:type="ns2:AttributedURI">http://10.100.64.65:8080/wsrf/services/sws/examples/BuyerSer
vice</ns2:Address>
 <ns2:ReferenceProperties xsi:type="ns2:ReferencePropertiesType">
```

```
  <ns1:BuyerResourceKey>7be97b40-162d-11db-9878-ce21816644d9</ns1:BuyerResourceKey>
 </ns2:ReferenceProperties>
 <ns2:ReferenceParameters xsi:type="ns2:ReferenceParametersType"/>
</ns1:BuyerServiceEndpoint>
Waiting for notification. Ctrl-C to end.
Waiting for notification. Ctrl-C to end.
```

10.8 Execution of Market Grid Service

```
$ java -DGLOBUS_LOCATION=$GLOBUS_LOCATION org.sws.examples.clients.GridManager.Client
http://10.100.64.65:8080/wsrf/services/sws/examples/GridManager market Kolkatta
```

10.9 Notification to Seller Service

```
Waiting for notification. Ctrl-C to end.

A new notification has arrived

Crop Price for potato is Rs60.0\kg

A new notification has arrived

Total Market Price for potatois Rs3600.0

A new notification has arrived

Vehicle Fees is calculated @ two rupee per unit of crop sold
Vehicle Fees charged for 60.0 Kg of crop is Rs. 120.0
Earning after deduction of vehicle fees is Rs3480.0

A new notification has arrived
Calculating Marketing Fees
Marketing Fees will be deducted
Marketing Fees Equal To:348.0

A new notification has arrived
Final Amount Offered to you is:Rs3132.0
```

11. Management of Grid Market

In the Grid environment, heterogeneous nature of resources, services and their functionality makes it difficult to manage them in a uniform way. Querying the functionality of any node and service is one of the hardest process and Grid management system often relies on the static WSDL documentation available to them even before the start interacting with the node, rather than determining its capabilities on the fly.

In our case study, their can be different type of markets specializing in different domains, sellers, trade scenarios and buyer interests etc. which makes the management very difficult if not possible. The situation becomes more complicated as the heterogeneity of the system can't be predicted at anytime and it should cope with any future situation. WSDM provides the uniform mechanism to interact with the heterogeneous resources for effective management. As discussed earlier WSDM specification describes what information any service or node may expose for uniform management. This information can be set of resource property elements, WSDM capabilities i.e. caption, description and version, manageability characteristics specific to its domain, operational status, metrics, notification of WSDM events, metadata for resource properties elements, relationship of different resources and resource properties elements.

Although we haven't discussed different type of markets and different trade scenarios in our case study, but still we have to monitor the activities of sellers and buyers. To manage different stake holders involves exposing new set of resource property elements and corresponding operations. The implementation of these manageable resources and operations is very similar to the any ordinary resource and has not discussed in the case study. The possible manageable

resource for the seller is discussed briefly, so that readers can appreciate the usage and role of WSDM in any Grid application. The possible resource properties elements for the seller manageable resource can be which can be modified only by the authentic management client:

- *Number of previous transactions:* This is metric metadata monitored for certain time period such as last 6 months. This resource property is updated for each successful transaction.
- *Number of items to be sold:* This is simple counter which monitors the item still available for sale.
- *Pending Fee:* This resource property is of data type boolean; which can have value false if the user has failed to pay any previous market fee for using any utility service.
- *Ranking:* This resource property indicates the ranking of the seller based on seller's previous activities i.e. successful transactions, activeness in the market, any complaints against the seller, his promptness to pay market fee etc.

Similarly their can be few management operations for the seller which depends on the manageable resource, update these resources, or take appropriate actions in case of any notification. Few of such operations are listed below:

- *suspendMembership:* This operation temporarily suspends the membership of the seller may be due to any on going dispute.
- *resumeMembership:* This operation lets seller to continue trading in the market once the issues related to its suspensions are resolved.
- *cancelMembership:* This operation cancels the membership of the seller permanently.
- *updateRanking:* After any transaction, sellers intentions to sell any new thing or any complaint against seller updateRanking operation is called automatically.

These manageable resources and corresponding operation are important for efficient queries. The buyer can search the seller based on the rankings, management system can offer incentives to the seller based on its activeness in market trading. This brings lot more confidence in the Grid Market as only management clients have complete access to these resource properties values and only few read-only resource properties are available to users of the system. On the same lines their will be manageable resource properties for the buyer and market.

12.Summary

Business processes of large enterprise systems interact with various stack holders i.e. business partners and customers to integrate distributed heterogeneous services. In this chapter, different specifications originating from the Grid paradigm are evaluated to capture the requirements of dynamic business process which is called Business Process Grid. These long running and dynamic business processes require support for state monitoring, transaction management, and event notifications.

We focused our attention towards the utilization and integration of existing technologies to achieve loosely coupled integration of services having support for state, notification, execution monitoring and scalability. Principles of Grid computing utilizing the Service Oriented Architecture (SOA) are followed and implemented to satisfy the requirements of the business process. To achieve these objectives, WS-RF, WSN, and relevant specifications and standards in this area are followed here. We have shown the use of WS-RF to achieve state in Web Services and WSN for notification and integration of distributed heterogeneous services. We have demonstrated the use of grid services to provide required middleware support. Convergence of grid and Web services standards and specifications for the complete execution of a business process is discussed in this chapter. The future work is diverted into several paths:

implementation of policy, agreement, negotiation, and use of semantic web to provide meaningful integration and coordination of resources in a grid environment.

13. References

Akram, A. Web Services Resource Framework Resource Sharing, Tutorial 9 and Tutorial 10, http://esc.dl.ac.uk/WOSE/tutorials/.

Birman, K. (2005). Can Web Services Scale Up? *Computer, 38*(10), 107-110.

Booth, D., Haas, H., McCabe, F., & et al. (2004). Web Services Architecture, W3C Working Group Note 11 February 2004. Available at http://www.w3.org/TR/ws-arch/.

Czajkowski, K., Ferguson, D., Foster, I., & et al. (2004). From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution.

Foster, I. (2002). *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, www.globus.org/alliance/publications/papers/ogsa.pdf.*

Foster, I., Czajkowski, K., & et al. (2005). *Modeling and managing State in distributed systems: the role of OGSI and WSRF.* Paper presented at the Proceedings of the IEEE.

Gottschalk, K. D., Graham, S., Kreger, H., & Snell, J. (2002). Introduction to Web services architecture. *IBM Systems Journal, 41*(2), 170-177.

Joseph, J., & Fellenstein, C. (2003). *Grid Computing*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Junginger, M. O. (2003). *High Performance Messaging System for Peer-to-Peer Networks.* University of Missoury, Kansas.

Laliwala, Z., Jain P. & Chaudhary S. (2006), *Semantic based Service-Oriented Grid Architecture for Business Processes*, 2006 IEEE International Conference on Services Computing (SCC 2006), SCC 2006 Industry track

Modal Act. (2003). The State Agricultural Produce Marketing (Development & Regulation Act, 2003), http://agmarknet.nic.in/reforms.htm.

Pallickara, S., & Fox, G. (2005). *An Analysis of Notification Related Specifications for Web/Grid Applications.* Paper presented at the ITCC (2).

Papazoglou, M. P. (2003). Web Services and Business Transactions. *World Wide Web, 6*(1), 49-91.

Schmit, B., & Dustdar, S. (2005). *Systematic Design of Web Service Transactions*.

SOAP. (2000). Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

Sorathia, V., Laliwala, Z., & Chaudhary, S. (2005). *Towards Agricultural Marketing Reforms: Web Services Orchestration Approach.* Paper presented at the SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing, Washington, DC, USA.

UDDI. (2004). Universal Description, Discovery and Integration, UDDI Version 3.0.2, http://uddi.org/pubs/uddi_v3.htm.

Vinoski, S. (2004a). More Web services notifications. *Internet Computing, IEEE, 8*(3), 90-93.

Vinoski, S. (2004b). WS-nonexistent standards. *Internet Computing, IEEE, 8*(6), 94-96.

Wang, H., Huang, J. Z., Qu, Y., & Xie, J. (2004). Web services: Problems and Future Directions. *Journal of Web Semantics, 1*(3).

Wilkes, L. (2005). The Web Services Protocol Stack, http://roadmap.cbdiforum.com/reports/protocols/index.php.

WS-Addressing. (2004). Web Services Addressing, W3C Member Submission 10 August 2004, http://www.w3.org/Submission/ws-addressing/.

WS-BaseFaults. (2006). Web Services Base Faults 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.

WS-BaseNotification. (2006). Web Services Base Notification 1.3, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-pr-03.pdf.

WS-BrokeredNotification. (2006). Web Services Brokered Notification 1.3, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-pr-03.pdf.

WS-Eventing. (2006). Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/WS-Eventing/

WS-Notification. (2005). Web Services Notification, www.oasis-open.org/committees/wsn/

WS-ResourceFramework. (2005). Web Services Resource Framework, www.oasis-open.org/committees/wsrf/

WS-ResourceLifetime. (2006). Web Services Resource Lifetime 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.

WS-ResourceProperties. (2006). Web Services Resource Properties 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

WS-ServiceGroup. (2006). Web Services Service Group 1.2, http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf.

WS-Topics. (2006). Web Services Topics 1.3, http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-pr-02.pdf.

WSDL. (2005). Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

Liang-Jie Zhang, Haifei Li, and Herman Lam (2004). Toward a Business Process Grid for Utility Computing. IT Professional, IEEE 6(5), 64 – 63.