SETIT 2007
4rth International Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications
March 25-29, 2007 – TUNISIA

Utilising Grid Services in ebXML Registry

Bahareh Rahmanzadeh Heravi, Mohammadreza Razzazi

Computer Engineering and Information Technology Department, Amirkabir University of Technology, Hafez Avenue, Tehran, Iran

rahmanzadeh@gmail.com

razzazi@ce.aut.ac.ir

Abstract: The aim of this paper is to demonstrate how to bring together two independent technologies, taking advantage of the complementarily suitable characteristics of each. The technologies are Grid Services and ebXML Registry. Specifically, the goal of this paper is to show how an ebXML Registry may be used to publish, store, manage and discover Grid Services. This will be valuable as a tool to enable trading partners to conduct business over a distributed environment using a standard framework, where the environment is the Grid and the standard framework is ebXML.

Key words: ebXML, ebXML Registry, Grid Services, Grid technology, Global E-marketplace.

INTRODUCTION

In recent years there has been an increasing awareness in the significance of the e-business and the numerous new opportunities it offers and the challenges that imposes. As a result, many companies around the world have decided to take advantage of this new phenomenon and start their own e-business. Because the cost of running an e-business is relatively high, small and medium size companies wouldn't be able to participate in such a market. Furthermore there are no standard protocol between them for their activities, from finding a trading partner through to making and closing a contract. In such conditions, the participants have to do with many inefficiencies and compromises. One solution is to have a global market, which all businesses can participate in forming an e-marketplace. For conducting business in such an e-marketplace we need a robust and standards-based environment as a framework. One such framework is ebXML (Electronic Business using eXtensible Markup Language).

ebXML is a modular suite of specifications that enables enterprises of any size to conduct business over the Internet from any geographical location. Using ebXML, companies now have a standard method to exchange business messages, establish trading relationships, communicate data in common terms and define and register business processes. [6]

Grid technologies, on the other hand, support the sharing and coordinated use of diverse resources in dynamic Virtual Organizations [15]. That is, the creation of virtual computing systems from geographically and organisationally distributed components. As the aim of both entities is to be independent of geographical locations, it will be desirable to utilise the grid as the environment for ebXML to form a global e-marketplace.

Grid technology uses web services on the Grid environment, called 'Grid Services', that is defined by OGSA (Open Grid Service Architecture) as a Web Service that provides a set of interfaces and behaviors that determines how a client interacts with a Grid service and conforms to a set of conventions [8].

Currently, Grid technology is mostly used in sciences. Considering the fact that ebXML is one of the most complete e-business frameworks, the convergence of ebXML and Grid technology could provide a wholesome framework for the future of e-business.

In an ideal global e-marketplace with shared resources and services, trading partners can understand each other without any previous conformity. It means that potential trading partners can join this e-marketplace and immediately start their business. Any business, regardless of its size, will have the opportunity to participate in the global market and take advantage of the opportunities that might arise.

In this paper we demonstrate how to publish, store, manage and discover OGSI (Open Grid Services

Infrastructure) Based Grid Services through ebXML Registry. Using Grid Services, we will be working with services that are already on the Grid environment, and using ebXML, we will take advantage of the e-business framework. Thus we will be able to offer Grid Services through a unified e-marketplace. Trading partners will be able to sell Grid Services as part of their services while they are in a unified structure at the same time.

1. Background

1.1. ebXML

ebXML was started in 1999 as an initiative of OASIS and the United Nations/ECE agency CEFACT. The original project envisioned and delivered five layers of substantive data specification, including XML standards for the following: [6]

- Registry/Repository
- Business Process Specification Schema
- Core Components
- Collaboration Protocol Profiles and Agreements (CPP/A)
- Messaging Service

In this paper we will only focus on ebXML Registry as the rest will remain unchanged for the purposes of our discussion.

1.2. ebXML Registry

The ebXML Registry is central to the ebXML architecture. An ebXML Registry provides a stable store where information submitted by an organization is made persistent. Such information is used to facilitate ebXML-based Business to Business (B2B) partnerships and transactions. Submitted contents may be XML schema and documents, process descriptions, Web Services, ebXML Core Components, context descriptions, UML models, information about parties and even software components. It enables secure and federated information management. Key ebXML Registry features are shown in figure 1 below: [1]

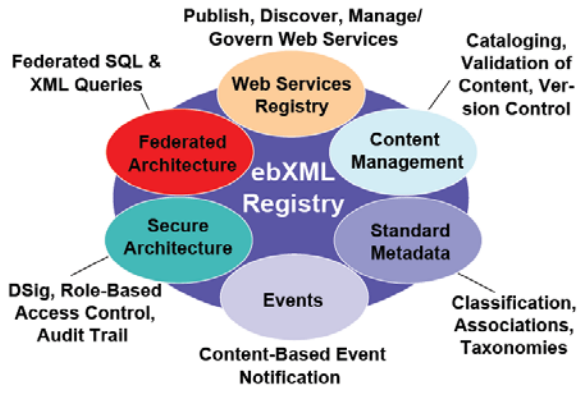

Figure 1. *Key ebXML Registry features [1]*

ebXML Registry can be used to register and discover Web Services. As Grid Services are a kind of Web Services, ebXML Registry may be used for registering and discovering Grid Services too. But this process requires some enhancements to be able to handle the differences between Grid Services and Web Services, on one hand and on the other hand take advantage of Grid technology's features.

1.3. Grid Computing

Grid computing is a way of organising computing resources so that they can be flexibly and dynamically allocated and accessed, often to solve problems requiring many organisations' resources. The objective of Grid computing is to make resources available so that they can be more efficiently utilised. [12]

The advantage of sharing is clearest when the need for resources is unpredictable, short-term, changes quickly, or where it is too large for any single organisation's capability to provide for it. For example, a problem that may take days to solve on a single installation's processing resources can be solved in a few minutes with the right kind of parallelization and distribution on additional allocated computational resources. [12]

1.4. Grid Services

Grid Services are an extension to Web Services for operating in Grid environments. The Open Grid Services Infrastructure (OGSI) provides an infrastructure layer for the OGSA to resolve the differences between Grid Services and Web Services.

1.4.1. Differences between Grid Services and Web Services

One extension to Web Services which Grid Services introduce is the Factory model. The Factory design pattern is commonly used in Object-Oriented software systems to enable the creation of multiple, similar artefacts. An OGSI Factory Service is a Grid Service that is used by a client to create other Grid Service Instances. When a client needs to create a new instance of a particular Grid Service, it locates a corresponding Factory Service, invokes its related operation, and receives a unique identifier that can be used to access the newly-created Instance [11]. The Factory will start a new Grid Service for that client. The generated Grid Service terminates after either a set time, or when requested to do so by the client.

Another extension to Web Services that a Grid Service introduces is Service Data. Service Data provides the information about a service, which the interface alone cannot provide. Every Grid Service has a basic set of Service Data Elements (SDEs), which contain information about the service, its interface and its location [7].

The Web Service interface is described by WSDL (Web Services Description language) whose function is to describe the service interface and enable clients to invoke the service. OGSI adds the features of Grid Services to basic Web Services by redefining the

WSDL portType element. Grid Services are described using an extended form of WSDL known as GWSDL (Grid WSDL) [12].

WSDL breaks down Web services into a number of elements, which are as follows: [16]

- *Service*

A collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

- *Port*

A combination of a binding and a network address, providing the target address of the service communication

- *Binding*

The concrete protocol and data formats for the operations and messages defined for a particular port type.

- *Port type*

An abstract set of operations mapped to one or more end points, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings.

- *Operation*

The abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.

- *Message*

An abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

- *Data types*

The data types—in the form of XML schemas or possibly some other mechanism—to be used in the messages

While a WSDL portType contains only operations, an OGSI portType can also contain SDEs. To all intents and purposes a SDE is a property or attribute of the Web Service (comparable to properties of classes and objects).

In OGSI, a portType can be constructed by referencing an existing portType or portTypes, with the extends attribute, adding new definitions as required. [12]

GWSDL also extends WSDL by introducing the concept of PortType inheritance. A PortType can inherit operations and Service Data Elements from other

2. Grid Services and ebXML Registry

Grid Services need a Registry to be published, managed and discovered. There are a few candidates for such a Registry. ebXML Registry / Repository is one of the leading Registries in the e-business world, that was accepted as a standard registry by ISO in March 2004 as ISO 15003 and ISO 15004. Using ebXML Registry we will be able to provide Grid Services in a unified, integrated B2B framework and use its advanced capabilities.

We need to find a method to take advantage of ebXML Registry for registering Grid Services. To achieve this, we map Grid Services Information Model onto the ebXML Registry Information Model (ebRIM). There is already a profile of ebXML Registry [3], which shows how to register Web Services in an ebXML Registry, and here we will demonstrate how to register Grid Services on the ebXML Registry.

3. Mapping Grid Services to ebXML Registry

3.1. Mapping overview

In order to register Grid Services into ebXML Registry, we will first map the GWSDL data structure to related ebXML Registry classes. In the Class diagrams below the GWSDL Information Model is shown on the left, and the mapping of this information model to ebRIM is shown on the right:

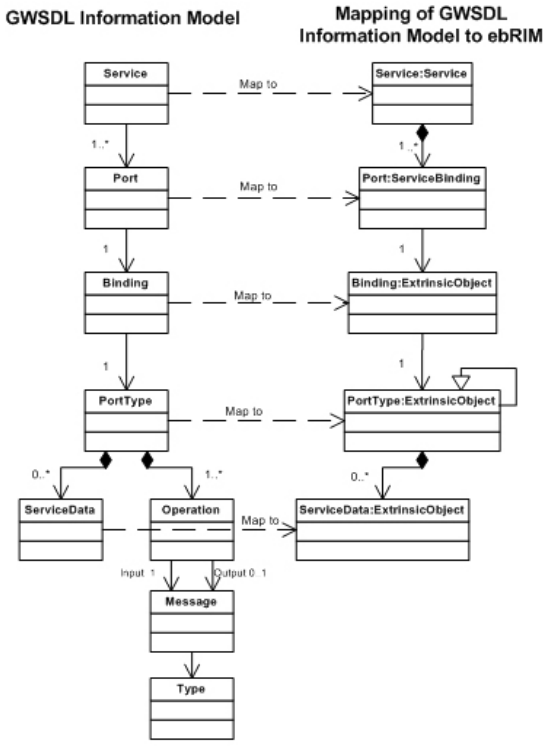

Figure 2. *Mapping of GWSDL Information Model entities to ebXML Registry Information Model classes*

As shown in figure 2, each Service entity in GWSDL must be mapped to a Service class in ebXML Registry. The Port entity must be mapped to a ServiceBinding class. Binding entity and PortType entity must be mapped to ExtrinsicObject classes [3]

which are the primary metadata class for a RepositoryItem. ServiceData entity should also be mapped to ExtrinsicObject classe. The rest of the GWSDL document does not need to be mapped.

3.1.1. Handling the differences between Grid Services and Web Services

ebXML Registry Information model is an object oriented model. Therefore for each service, an instance will be created. In this way, we can resolve the first of the three differences between Web Services and Grid Services which is the Factories concept (see 1.4.1 above). Factory's responsibility is to create different instances of Services. Here each Service is an ebXML Registry Class and consequently it can have as many instances.

As mentioned in 1.4.1, another difference between Grid Services and Web Services is Service Data. To overcome this, we have added a class called ServiceData to ebRIM. This class is an ExtrinsicObject that is the same as binding and portType. Each PortType can have zero or more ServiceData. In this case we can manage the ServiceData extension to Web Services. We can also use them to search in the ebXML Registry which will be described later.

Finally, as shown in figure 2, a PortType can inherit another PortType. By adding this relationship we overcome the third difference which is PortType inheritance.

3.2. Mapping detail

The rest of this paper is dedicated to the details of this mapping.

Firstly, we will compare a WSDL code with its equivalent GWSDL code to demonstrate the differences between the two that were described above. The codes only show the PortType and its lower levels, since the upper levels are the same in both WSDL and GWSDL.

```
<wsdl:portType name="PortTypeName">
  <wsdl:documentation>
    Port Type documentation
  </wsdl:documentation>
  <wsdl:operation name="opName">
    <wsdl:input message="xxxx"/>
    <wsdl:output message="xxxx"/>
  </wsdl:operation>
</wsdl:portType>
```

Listing 1. *wsdl:portType example code*

```
<ogsi:portType name=" PortTypeName"
extends="ogsi:GridService">
  <wsdl:documentation>
    Port Type documentation
  </wsdl:documentation>
  <wsdl:operation name=" opName">
    <wsdl:input message="xxxx"/>
    <wsdl:output message="xxxx"/>
  </wsdl:operation>
  <sd:serviceData name="counterValue"
                  type="xs:positiveInteger"
                  minOccurs="1"
                  maxOccurs="1"
                  mutability="mutable">
    <sd:documentation>
      The value of the counter.
    </sd:documentation>
  </sd:serviceData>
</ogsi:portType>
```

Listing 2. *ogsi:portType example code*

To achieve our goal we must map GWSDL entities' attributes to ebXML Registry Classes' attributes. This mapping is similar to the ebXML Registry profile for Web Services [3]. As that document describes the mapping between WSDL Information Model and ebXML Registry Information Model, we won't talk about the parts which are similar to WSDL. Instead we will focus on the Grid Services extensions to Web Services, as described earlier. However, in order to become familiar with the concept we will show a summary of the mapping of WSDL based entities to equivalent ebXML code. This mapping just shows the three top layers of WSDL and GWSDL data structure, which are the same. In the rest of the paper, mappings will only relate to GWSDL extensions to WSDL.

3.2.1. WSDL data structure mapping to ebRIM

The summary mapping of WSDL Information Model to ebRIM is shown in the listing 4. This listing may seem complicated, but this is a good way to show a summary of the mapping. For more information please refer to [3]. The mapped values are shown in *italic* and WSDL and ebRIM attributes are shown in **bold**. The listings 3 and 4 show this mapping up to, and including, the Binding entity of WSDL. As the PortType must be redefined for our purpose, we will follow the description of the rest of the GWSDL information model after the following coding in 3.2.2.

```
<!-- -------------- WSDL Service ------------ -->
<wsdl:service name="counterService">
  <wsdl:documentation>
    An implementation of counter Service
  <wsdl:documentation>

<!-- ------------------ WSDL Port -------------- -->
  < wsdl:port binding="bindings:counterBinding"
name="counterPort">
      <soap:address
location="http://your.server.com/counterService"/>
  </port>
<!-- ---------------- WSDL Port  end----------- -->

</service>
<!-- ------------ WSDL Service end----------- -->

<!-- ---------------- WSDL Binding ------------ -->
<binding name="counterBinding"
type="interfaces:counterPortType">
  ….
```

```
</wsdl:binding>
<!-- ----------- WSDL Binding end------------ -->
```

Listing 3. *The outline of three top level WSDL entities*

```
<!-- -------------- ebRIM Service ------------ -->
<rim:Service
id="urn:acmeinc:ebxml:registry:3.0:services:wsdl:service:
counterService">
   <rim:Name>
      <rim:LocalizedString value="counterService"/>
   </rim:Name>
   <rim:Description>
      <rim:LocalizedString value=" An implementation of
counter Service"/>
   </rim:Description>

<!-- ebRIM Port:ServiceBinding(WSDL Port mapping) -->

   <rim:ServiceBinding
id="urn:acmeinc:ebxml:registry:3.0:services:wsdl:port:co
unterrPort"
accessURI="http://your.server.com/counterService">
      <rim:Name>
         <rim:LocalizedString value="counterPort"/>
      </rim:Name>

<!-- ---- ebRIM Port:ServiceBinding end ----- ->

</rim:Service>
<!-- ------------ ebRIM Service end------------ -->

<!-- ebRIM Binding: ExtrinsicObject (WSDL Binding
mapping) -->

<rim:ExtrinsicObject
objectType="urn:oasis:names:tc:ebxmlregrep:
ObjectType:RegistryObject:ExtrinsicObject:WSDL:Bindin
g"
id="urn:acmeinc:ebxml:registry:3.0:ExtrinsicObject:wsdl:
binding:counterBinding">
   <rim:Name>
      <rim:LocalizedString value="counterBinding"/>
   </rim:Name>
</rim: ExtrinsicObject>
<!-- ----- ebRIM Binding: ExtrinsicObject end ------- -->
```

Listing 4. *Mapping of Listing 3 code to ebRIM*

3.2.2. GWSDL data structure mapping to ebRIM

As discussed earlier, OGSI adds the features of Grid Services to basic Web Services by redefining the WSDL PortType element. In this part we show how to apply GWSDL extensions to ebXML Registry.

The first step is to redefine the PortType. As shown in the listing 2, which is a Grid Service's GWSDL code, we can see that a 'wsdl' namespace is replaced by 'ogsi' in the beginning of the portType tag. This shows that the PortType is a Grid Service PortType and not a Web Service one. Now we must establish a method to map this ogsi:portType and its attributes to ebXML Registry classes.

3.2.2.1. ogsi:portType to rim:ExtrinsicObject Mapping

In the GWSDL portType mapping to ebRIM a PortType instance must be mapped to a rim:ExtrinsicObject instance. In this process GWSDL PortType's attributes must be mapped to rim:ExtrinsicObject attributes as described below.

- *Attribute objectType*

The objectType attribute value of the rim:ExtrinsicObject must be set to urn:oasis:names:tc:ebxmlregrep: ObjectType:RegistryObject:ExtrinsicObject:OGSI:PortType.

```
<rim:ExtrinsicObject
   objectType="urn:oasis:names:tc:ebxmlregrep:
ObjectType:RegistryObject:ExtrinsicObject:OGSI:PortT
ype" ...>
```

Listing 5. *Example of rim:ExtrinsicObject objectType Attribute Mapping for ogsi:portType*

- *Attribute id*

The id attribute value of the rim:ExtrinsicObject must have as prefix the targetNamespace of the ogsi:portType element, followed by a suffix of ":portType:<portType name>" where <portType name> must be the value of the name attribute of the <ogsi:portType> element.

```
TargetNameSpace=
urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interfaces
:3.0
<ogsi:portType name="counterPortType"/>


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:
ogsi:registry:interfaces:3.0:portType:counterPortType"
...>
```

Listing 6. *Example of rim:ExtrinsicObject id Attribute Mapping for ogsi:portType*

- *Element Name*

The name element of the rim:ExtrinsicObject must be set according to the value of the name attribute within the ogsi:portType element.

```
<ogsi:portType name="counterPortType">


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interf
aces:3.0:portType:counterPortType">
   <rim:Name>
      <rim:LocalizedString value="counterPortType"/>
   </rim:Name>
</rim:ExtrinsicObject>
```

Listing 7. *Example of rim:ExtrinsicObject name Attribute Mapping for wsdl:portType*

- *Element Description*

The description element of the rim:ExtrinsicObject must be set according to the content of the

wsdl:documentation element within the ogsi:port element, if specified.

```
<ogsi:portType name="counterPortType">
   <wsdl:documentation>
      portType for Counter Grid Service
   <wsdl:documentation>
</ ogsi:portType>


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interfa
ces:3.0:portType:counterPortType">
   <rim:Description>
      <rim:LocalizedString value=" portType for Counter
Grid Service"/>
   </rim:Description>
</rim:ExtrinsicObject>
```

Listing 8. *Example of rim:ExtrinsicObject description Attribute Mapping for wsdl:portType*

3.2.2.2. ogsi:serviceData to rim:ExtrinsicObject Mapping

So far we have added a ServiceData Class to our ebXML Registry Information Model for modeling the equivalent entity in GWSDL Information Model. In the GWSDL ServiceData mapping to ebRIM, a ServiceData instance must be mapped to a rim:ExtrinsicObject instance. The details of this mapping are as follows.

- *Attribute objectType*

The objectType attribute value of the ServiceData class must be set to urn:oasis:names:tc:ebxmlregrep:ObjectType:RegistryObject:ExtrinsicObject:OGSI:ServiceData.

```
<rim:ExtrinsicObject
   objectType="urn:oasis:names:tc:ebxmlregrep:
ObjectType:RegistryObject:ExtrinsicObject:OGSI:Servi
ceData" ...>
```

Listing 9. *Example of rim:ExtrinsicObject objectType Attribute Mapping for ogsi:serviceData*

- *Attribute id*

The id attribute value of the ServiceData class must have as prefix the targetNamespace of the sd:serviceData element, followed by a suffix of "serviceData:<serviceData name>" where <serviceData name> must be the value of the name attribute of the <sd:serviceData> element.

```
TargetNameSpace=
urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interfaces
:3.0
<sd:serviceData name="counterValue"/>


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interf
aces:3.0: serviceData:counterValue" ...>
```

Listing 10. *Example of rim:ExtrinsicObject id Attribute Mapping for ogsi:serviceData*

- *Element Name*

The name element of the ServiceData class must be set according to the value of the name attribute within the ogsi:portType element.

```
< sd:serviceData name="counterValue">


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interf
aces:3.0:serviceData:counterValue">
   <rim:Name>
      <rim:LocalizedString value=" counterValue"/>
   </rim:Name>
</rim:ExtrinsicObject>
```

Listing 11. *Example of rim:ExtrinsicObject nameAttribute Mapping for ogsi:serviceData*

- *Element Description*

The description element of ServiceData class must be set according to the content of the sd:documentation element within the sd:serviceData element, if specified.

```
<sd:serviceData name="counterValue">
   <sd:documentation>
      documentation for Service Data
   <sd:documentation>
</wsdl:portType>


<rim:ExtrinsicObject
id="urn:oasis:names:tc:ebxmlregrep:ogsi:registry:interf
aces:3.0: serviceData:counterValue">
   <rim:Description>
      <rim:LocalizedString value=" documentation for
Service Data"/>
   </rim:Description>
</rim:ExtrinsicObject>
```

Listing 12. *Example of rim:ExtrinsicObject description Attribute Mapping for wsdl:portType*

4. Discovery of registered Grid Services

The Query Management protocols of ebXML Registry provide the functionality required by RegistryClients to query the registry and discover RegistryObjects and RepositoryItems. [4]

The Ad hoc Query protocol of the QueryManager service interface allows a client to query the registry and retrieve RegistryObjects and / or RepositoryItems that match the specified query. [4]

The AdhocQuery protocol allows clients to submit queries that may be as general or as complex. As the queries get more specific they also get more complex. In these situations it is desirable to hide the complexity of the query from the client using parameterized queries stored in the registry. When using **parameterized stored queries** the client is only required to specify the identity of the query and the parameters for the query rather than the query expression itself. [3]

When submitting a stored query, the submitter may declare zero or more parameters for that query. A parameter must be declared using a parameter name that begins with the '$' character followed immediately by a letter and then followed by any combination of letters and numbers. [4]

A stored query may be defined with zero or more parameters. A client may specify zero or more of the parameters defined for the stored query when submitting the AdhocQueryRequest for the stored query. [4]

A client specifies a query invocation parameter by using a **Slot** whose name matches the parameter name and whose value must be a single value that matches the specified value for the parameter. [4]

The listing 13 shows an example of how to invoke a stored query, where its name contains the string "ebXML".

```
<AdhocQueryRequest>
  <rim:AdhocQuery id="${STORED_QUERY_ID}">
    <rim:Slot name="$name">
      <rim:ValueList>
        <rim:Value>%ebXML%</rim:Value>
      </rim:ValueList>
    </rim:Slot>
  </rim:AdhocQuery>
</AdhocQueryRequest>
```

Listing 13. *Example of an AdhocQueryRequest*

The discovery queries are specified from the bottom of the layered GWSDL information model to the top. The query for each layer specifies parameters specific to it as well as parameters specific to each of the lower layers that it builds upon. Thus the number of parameters increases as queries are defined for higher level types in the model. This is key to being able to discover higher level objects based on attributes of the lower level objects that they build upon. [3]

There are many parameters supported for the discovery query for each higher level type in the model. However, it is often the case that discovery may not require parameters specific to all lower level types. To facilitate pruning of the discovery query for unwanted predicates related to lower level types there is a special parameter name $considerXXX where XXX represents a lower level type within the model. If the value of this parameter is set to "0" then all parameter values specific to lower level type must be ignored by the discovery query. [3]

4.1. GWSDL Document Discovery Query

The GWSDL Document discovery query must be implemented by an ebXML Registry implementing this profile. It allows the discovery of GWSDL documents using zero or more of the parameters described next.

- *Parameter $name*
  This parameter's value may specify a string containing a pattern to match against the name attribute value of RegistryObjects that have objectType of GWSDL.

- *Parameter $description*
  This parameter's value may specify a string containing a pattern to match against the description attribute value of RegistryObjects that have objectType of GWSDL.

- *Parameter $targetNamespace*
  This parameter's value may specify a string containing a pattern to match against the targetNamespace of a GWSDL document.

- *Parameter $importedNamespace*
  This parameter's value may specify a string containing a pattern to match against the namespaces imported by a GWSDL document.

4.2. ServiceData Discovery Query

The OGSI ServiceData discovery query allows the discovery of sd:serviceData instances using zero or more of the parameters described next.

- *Parameter $serviceData.name*
  This parameter's value may specify a string containing a pattern to match against the name attribute value of sd:serviceData instances.

- *Parameter $ serviceData.description*
  This parameter's value may specify a string containing a pattern to match against the description attribute value of sd:documentation instances.

The listing 14 shows an example of how to find all sd:serviceData instances that have a name containing the string "counterValue".

```
<AdhocQueryRequest>
  <rim:AdhocQuery id="${STORED_QUERY_ID}">
    <rim:Slot name="$serviceData.name">
      <rim:ValueList>
        <rim:Value>%counterValue%</rim:Value>
      </rim:ValueList>
    </rim:Slot>
  </rim:AdhocQuery>
</AdhocQueryRequest>
```

Listing 14. *Example of ServiceData Discovery Query*

4.3. PortType Discovery Query

The OGSI PortType discovery query allows the discovery of ogsi:portType instances using zero or more of the parameters described next.

- *Parameter $portType.name*
  This parameter's value may specify a string containing a pattern to match against the name attribute value of ogsi:portType instances.

- *Parameter $ portType.description*
  This parameter's value may specify a string containing a pattern to match against the description attribute value of ogsi:portType instances.

- *Parameter $portType.targetNamespace*

This parameter's value may specify a string containing a pattern to match against the targetNamespace of ogsi:portType instances.

- *Parameter $portType.schemaNamespaces*

This parameter's value may specify a string containing a pattern to match against the XML schema namespaces used within the wsdl:message instances used within the wsdl:operation instances used within the ogsi:portType instances.

- *Parameter $considerServiceData*

This is a $considerXXX parameter, where the XXX is ServiceData, facilitating the discovery of a PortType which has a ServiceData element with specific parameters. This parameter's value may specify a string of "1" or "0" to indicate whether or not to consider the ServiceData specific parameters that follow when processing the query. If unspecified the value defaults to "0".

- *Parameter $serviceData.name*

This parameter's value may specify a string containing a pattern to match against the name attribute value of sd:serviceData instances that are used by the objects being discovered.

- *Parameter $serviceData.description*

This parameter's value may specify a string containing a pattern to match against the content of the sd:documentation element within sd:serviceData instances that are used by the objects being discovered.

4.4. Service, Port and Binding Discovery Queries

The Port and Binding discovery queries allow the discovery of wsdl:port and wsdl:binding instances respectively, using their parameters. In the same way, the Service discovery query allows the discovery of wsdl:service/ogsi:GridService instances using their parameters. These three top level entities in the GWSDL Information Model are the same as their equivalents in WSDL Information Model. The main parameters of these entities are described in the ebXML Registry profile for Web Services [3].

The only things that will have to be added to these parameters are related parameters of each entity's ServiceData as their lower layer parameters. These parameters work in the same way in all three entities and enable them to consider ServiceData's parameters as their own lower layer parameters. This is exactly the same as the PortTye discovery query for searching on the ServiceData's parameters, which is discussed in 4.3. To achieve this, we suggest adding three parameters to those of each entity's. The first Parameter is $considerServiceData, which enables us to consider ServiceData's parameters in our discovery queries. The other two are $serviceData.name and $serviceData.description. The details of these parameters are similar to those of the PortType's, which is described in 4.3. Using these parameters, we will be able to discover Service, Binding and Port instances, which have a ServiceData with specific parameters.

For example, the Service entity, which is the top most level entity, must have four $considerXXX parameters. These are $considerBinding, $considerPort, $considerPortType and the newly added $considerServiceData plus their related parameters such as $binding.name or $serviceData.description.

The listing 15 shows an example of how to find all Service instances with the name "counterService", which have a portType with the name "counterPortType" and a serviceData with the name "counterValue".

```
<AdhocQueryRequest>
  <rim:AdhocQuery id="${STORED_QUERY_ID}">
    <rim:Slot name="$service.name">
      <rim:ValueList>
        <rim:Value>counterService</rim:Value>
      </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="$considerPortType">
      <rim:ValueList>
        <rim:Value>1</rim:Value>
      </rim:ValueList>
    </rim:Slot>
    </rim:Slot>
    <rim:Slot name="$portType.name">
      <rim:ValueList>
        <rim:Value>counterPortType</rim:Value>
      </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="$considerServiceData">
      <rim:ValueList>
        <rim:Value>1</rim:Value>
      </rim:ValueList>
    </rim:Slot>
    <rim:Slot name="$serviceData.name">
      <rim:ValueList>
        <rim:Value>counterValue</rim:Value>
      </rim:ValueList>
    </rim:Slot>
  </rim:AdhocQuery>
</AdhocQueryRequest>
```

Listing 15. *Example of Service Discovery Query*

5. Conclusion

In our earlier paper [10] we demonstrated a way to do business using ebXML standards on the Grid environment using UDDI registry. Here we have shown how to bypass UDDI and take advantage of, the more sophisticated, ebXML Registry. It is important to note that the aim is not to provide a set of instructions, but to give a method to achieve this goal.

ACKNOWLEDGMENT

I would like to express my gratitude and appreciation to Farrukh Najmi for his guidance and kind support. Farrukh Najmi is the main author of ebXML Registry specification, founder and lead architect of freebXML Registry open source project, principal architect of Sun Service Registry product

and specification lead of Java API for XML Registries.

This paper is sponsored by Iran Telecommunication Research Centre, Ministry of Information & Communication Technology.

REFERENCES

[1] ebXML Registry 3.0: An overview, OASIS, http://www.oasis-open.org/committees/download.php/13010/ebXMLRegistryOverview.pdf.

[2] ebXML Registry Information Model, V3.0, Sally Fuger, Farrukh Najmi, Nikola Stojanovic, OASIS, May 2005, http://docs.oasis-open.org/regrep-rim/v3.0/.

[3] ebXML Registry profile for Web Services, Version 1.0 Draft 3, Farrukh Najmi, Joseph Chiusano, OASIS, September 2005, http://ebxmlrr.sourceforge.net/tmp/regrep-ws-profile-1.0.pdf

[4] ebXML, "OASIS/ebXML Registry Services Specification", V3.0, , Sally Fuger, Farrukh Najmi, Nikola Stojanovic, OASIS, May 2005, http://docs.oasis-open.org/regrep-rs/v3.0.

[5] ebXML Technical Architecture Specification, v1.0.4, Anders Grangard, Brian Eisenberg, Duane Nickull, Colin Barham, …, 2001, http://www.ebxml.org/specs/ebTA.pdf.

[6] ebXML, http://www.ebXML.org

[7] Enhancing UDDI for Grid Service Discovery by Using Dynamic Parameters, ICCSA 2005, Springer-Verlag Berlin Heidelberg, 2005.

[8] *Grid Service Specification*, Steven Tuecke, Karl Czajkowski, Ian Foster, Jeffrey Frey, Steve Graham, Carl Kesselman, Global Grid Forum, 2002.

[9] *Grid Computing: A Practical Guide to Technology and Applications*, Ahmar Abbas, Firewall Media, 2004.

[10] *Grid based Specifications of eXML*, Bahareh Rahmanzadeh Heravi, Mohammadreza Razzazi, 2nd IEEE International Conference on Information & Communication Technologies: From Theory to Applications, April 2006.

[11] *Grid Computing: A Practical Guide to Technology and Applications*, Ahmar Abbas, Firewall Media, 2004.

[12] Open Grid Service Infrastructure Primer, Global Grid Forum, August 2004, http://www.ggf.org/ogsi-wg.

[13] Open Grid Services Infrastructure (OGSI), Version 1.0, S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, ..., June 2003, http://www.ggf.org/ogsi-wg.

[14] *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Foster, I., Kesselman, C. and Tuecke, S. International Journal of High Performance Computing Applications, 2001, http://www.globus.org/research/papers/anatomy.pdf.

[15] The Open Grid Services Architecture, Version 1.5, I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, …, 24 July 2006, http://forge.gridforum.org/projects/ogsa-wg

[16] *Understanding Web Services, XML, WSDL, SOAP and UDDI*, Eric Newcomer, Addison Wesley, 2002.