

Category: INFORMATIONAL

May, 27th, 2005

Web Services Data Access and Integration – The Object Realisation (WS-DAIO)

Status of This Memo

This memo provides information regarding the specification of service-based interfaces to object data resources. The specification is presently a draft for discussion. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Copyright © Global Grid Forum (2005). All Rights Reserved.

1 Abstract

Data resources play a significant role in many applications across multiple domains. Web services provide implementation neutral facilities for describing, invoking and orchestrating collections of networked resources. The GGF (Global Grid Forum) Open Grid Services Architecture (OGSA), and its associated specifications, defines consistent interfaces through web services to components of the grid infrastructure. Both the web and grid communities stand to benefit from the provision of consistent and agreed web service interfaces for data resources and the systems that manage them.

This document, *Web Services Data Access and Integration: The Object Realisation (WS-DAIO)*, presents a specification for a collection of data access interfaces for object data resources, which extends interfaces defined in the *Web Services Data Access and Integration* document [WS-DAI].

Related DAIS specifications define how other data resources and systems can be described and manipulated through web services. The DAIS specifications form part of a broader activity within the GGF to develop OGSA. The DAIS specifications can be applied in regular web services environments or as part of a grid fabric.

31 Contents

32

1 Abstract.....	1
2 Introduction.....	3
2.1 Specification Scope.....	3
2.2 Specification Organisation.....	3
2.3 Interface Composition	3
3 Notational Conventions	4
4 Terminology	4
5 Concepts.....	5
5.1 Port Types	5
5.1.1 Data Description.....	5
5.1.2 Data Access	5
5.1.3 Data Factory.....	5
5.2 Relationships with other specifications.....	5
6 ObjQL	7
6.1 ObjAccessDescription	7
6.1.1 QueryLanguage.....	7
6.2 ObjAccess	7
7 ObjResponse.....	8
8 ObjList.....	8
Intellectual Property Statement	9
Full Copyright Notice	9

33

34

35

36

37

2 Introduction

Data access plays a central role for many types of Grid applications. Data access generally involves the retrieval, manipulation and insertion of data, which may be stored using a range of different formats and infrastructures.

This document presents a specification for a collection of data access interfaces for relational data resources. A relational data resource is a data source/sink, together with any associated management infrastructure, that is characteristic of relational database systems, e.g., can be queried or updated using SQL or any other suitable relational query/update language. The Port Types are thus categorized according to the support they provide for:

- Data Description
- Data Access
- Data Factories

As such, this document should be read in conjunction with the generic *Web Services Data Access and Integration* document [WS-DAI], which defines the base interfaces that are extended in this document. These specifications are being developed for representing data resources as web services, and form part of a broader activity within the Global Grid Forum to develop the Open Grid Services Architecture (OGSA) [OGSA].

2.1 Specification Scope

The DataAccess, DataFactory and DataManagement interfaces and the role of Data Description in the provision of service-based interfaces to data resources are discussed in the *Web Services Data Access and Integration* document [WS-DAI]. This specification extends those interfaces to allow access to and description of relational data resources. The relational data resources are assumed to be composed of tabular data structures such as relations and result sets typically accessed either using SQL queries or by row iteration.

2.2 Specification Organisation

This specification separates the function of a Data Service from its operational representation as expressed in WSDL. This approach allows functions to be mapped to WSDL in more than one way and a single such mapping is provided in Section 9.

The relational model is described using the terminology defined in Section 4 and employs the concepts described in Section **Error! Reference source not found..** Sections 5, 6 and 7 describe the interfaces for posing SQL queries, accessing the results of SQL queries and for iterating through result sets respectively

An illustrative mapping of the relational model to the Web Services Resource Framework (WSRF) [WS-Resource] is described in Section 8, Section 9 discusses security and Section 10 draws conclusions from this specification exercise.

2.3 Interface Composition

This specification does not mandate how interfaces are composed into services; the proposed interfaces may be used in isolation or in conjunction with others. Viable compositions of interfaces will, initially, follow established patterns for data access.

Here a Data Service provides SQLAccess, SQLResponseAccess and SQLRowSetAccess interfaces for a relational Data Service that is associated with a relational database.

3 Notational Conventions

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC-2119 [RFC2199].

When describing concrete XML schemas, this specification uses the notational convention of [WS-Security]. Specifically, each member of an element's children or attributes property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1 indicates that namespace x is being used, the root element *MyHeader* and a child element *SomeProperty* with an attribute *value1*). The use of {any} indicates the presence of an element wildcard (<xsd:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xsd:anyAttribute/>).

When patterns of messages are described the layout of the XML of each message is presented, as opposed to the XML schema. The following notation is used to indicate cardinality of XML elements in these cases:

- * zero or more
- + one or more
- ? zero or one

Where no notation is added to an element only one instance of the element is expected.

This specification uses namespace prefixes throughout; these are listed in the table below. Note that the choice of any namespace prefix is arbitrary and is not semantically significant.

Prefix	Namespace
http	<u>http://www.w3.org/2002/06/wsd/</u>
wsdl	<u>http://schemas.xmlsoap.org/wsdl/</u>
xsd	<u>http://www.w3.org/2001/XMLSchema</u>
wsdai	<u>http://www.ggf.org/namespaces/2004/09/WS-DAI</u>
wsdair	<u>http://www.ggf.org/namespaces/2004/09/WS-DAIR</u>
wsdaisr	<u>http://www.ggf.org/namespaces/2004/09/WS-DAISR</u>
wsdairs	<u>http://www.ggf.org/namespaces/2004/09/WS-DAIRS</u>
wsdaio	<u>http://www.ggf.org/namespaces/2005/05/WS-DAIO</u>
wrs	<u>http://java.sun.com/xml/ns/jdbc/webrowset.xsd</u>
wsa	<u>http://schemas.xmlsoap.org/ws/2004/03/addressing</u>

4 Terminology

The model independent terminology, i.e., data resource, Data Service, Consumer and Data Set, is given in the *Web Services Data Access and Integration* document [WS-DAI].

The model independent terminology, i.e., data resource, Data Service, Consumer and Data Set, is given in the *Web Services Data Access and Integration* document [WS-DAI].

5 Concepts

5.1 Port Types

DAIS classes its Port Types into three broad categories, which are defined in the WS-DAI specification. They are extended in this document to target relational data resources.

5.1.1 Data Description

The *DataDescription* interfaces allow a description of data represented by Data Services to be provided. The model-independent specification for these is given in the *Web Services Data Access and Integration* document [WS-DAI]. Here they are extended to provide a description of relational data resources. These are the main points of extension for object data resources:

- *ObjAccessDescription*
- *ObjResponseDescription*
- *ObjListDescription* provides information about a particular instance of a query result that a Data Service may represent. This interface will make available information about the schema for representing the query result and the number elements within the *ObjList*.

These capabilities are described in Sections .

5.1.2 Data Access

DataAccess operations allow object data resources to be modified through insertion, updates or deletes, or queried through an appropriate language. Some object data resource products also support XML access – these are addressed in the WS-DAIX specification. The following Data Access interfaces are defined in this specification:

- *ObjAccess*: provides access to a relational data resource.
- *ObjResponseAccess*: provides access to each type of Response that can result from the execution of a *ObjQLEExpression*.
- *ObjListAccess*: provides access to a set of rows, which are usually the result of a *ObjQLEExpression* containing a SELECT statement.

These are covered in more detail in Sections **Error! Reference source not found.** to **Error! Reference source not found.** .

5.1.3 Data Factory

The *DataFactory* interfaces allow data represented in relational data resources, usually as the result of a query or update, to be instantiated as Data Services. The specializations in this instance thus deal with the type of *ObjQLEExpression* that can be passed to a *DataFactory* to expose the results in a meaningful fashion. The properties and interfaces that will be supported by these Data Services are specified in the schema for the creation parameters. *DataFactory* specializations are:

- *ObjAccessFactory*: provides access to a object data resource.
- *ObjResponseFactory*: provides access to a response of an object query.

These are covered in more detail in Sections **Error! Reference source not found.** to **Error! Reference source not found.**

5.2 Relationships with other specifications

WS-DAIO does not provide its own query/update languages for object data resources. Instead, it acts as a conduit for existing object query and update languages to be conveyed to the appropriate data resources, in this instance relational data resources or a data resource that supports relational type queries. As such WS-DAIO relies on existing object query and update languages. In this document, interface support is provided for languages based on the following standards:

- Hibernate QL, the query language for the Java centric O/R persistence model Hibernate which is seen as the basis for the upcoming EJB v3 Entity Bean specification [Hibernate]
- EJB-QL versions 2 and 3, the query languages for Entity Enterprise Java Beans (EJB) versions 2 (as defined in J2EE v1.4) and 3 (upcoming, likely J2EE v1.5) [EJB2, EJB3]
- JDO-QL, the query language of the Java Data Object (JDO) specification JSR 12 [JSR12]
- XQuery, the XML query language [XQuery]
- Custom QL can be added using a different name space

[

Norman: JD-OQL as one of the query languages. I am conscious that JDO-QL queries are normally represented as Java objects, so it needs to be stated how these are represented as parameters to a message. If this was an XML structure, then it may be quite straightforward to support this over existing platforms that don't support JDO-QL directly.

]

6 ObjQL

6.1 ObjAccessDescription

In the contrary to access to relational databases where a sophisticated standardized query language exists (SQL), there is no single counterpart to it in the world of object data bases or O/R mapped relation databases (both referred to as object resources in this document).

In the contrary, currently several standards and quasi standards exist which are broadly applied. So a specification of WS based access to object resources such as WS-DAIO can either provide access employing a new query language made up of the most prominent ones or support the use of the "native" query languages themselves. The authors of this specification have chosen to follow the second approach.

Furthermore, the queries to the object resource are in some query language to be formulated at compile- or deploy-time. That means, that a query to an object resource cannot be freely formulated by the client but rather has to be based on the available (implemented and/or deployed) query methods provided by the object resource (comparable to a stored procedure). The particularities of each of the default query language implementation are discussed in the according subsection later within this section.

[

This subsection is intended to describe the access description of object resources.

It has to be thought how the different types of storages which have to be represented. A first draft should contain hibernate, since it a feasibility study can easily be based on hibernate. EJB3 is still a draft itself but will be quite similar to hibernate. JDO integration should be also quite straight forward, which EJB2 may be more complex due to its internal structure. Other proprietary query languages should be integratable as well.

XQuery support should be written in collaboration with DAIX.

Do we need to specify more than the QueryLanguage?

]

6.1.1 QueryLanguage

WS-DAIO directly supports by default the following query languages:¹

Query Language	Version
EJB-QL	2.1 (CMP EJB, J2EE 1.4) 3 (JSR 220)
Hibernate	3
JDO	1.0.1 and 2 (JSR 22)
XQuery	1.0 (in coordination with WS-DAIX)

Furthermore, WS-DAIO is open to custom extensions, so that e.g. the Oracle TopLink QL can be added using a different name space.

The following part of the schema definition accommodates the support for default and custom query languages.

¹Should there be support for OQL and/or SQL++, too? -> Leon, can you answer this?

```

230 <xsd:simpleType name="QueryLanguageEnumeration">
231   <xsd:union>
232     <xsd:restriction base="xsd:string">
233       <xsd:enumeration value="HibernateQL"/>
234       <xsd:enumeration value="EJB-QL"/>
235       <xsd:enumeration value="JDO"/>
236       <xsd:enumeration value="XQuery"/>
237     </xsd:restriction>
238     <xsd:any namespace="##other"/>
239   </xsd:union>
240 </xsd:simpleType>
241
242 <xsd:simpleType name="version">
243   <xsd:restriction base="xsd:string">
244     <xsd:pattern="([0-9])+(\.([0-9])+)*/>
245   </xsd:restriction>
246 </xsd:simpleType>
247
248 <xsd:element name="QueryLanguage">
249   <xsd:attribute name="language"
250     type="wsdaio:QueryLanguageEnumeration"/>
251   <xsd:attribute name="version" type="wsdaio:version"
252     minOccurs="0"/>
253   <xsd:anyAttribute namespace="##other" processContents="lax"/>
254 </xsd:element>

```

6.1.2 EJB-QL

[This subsection or parts of it may go to an annex]

Queries in EJB-QL 2.1 are static. They have to be formulated at compile/deploy time and hence this EJB CMP persistence does only allow access to special methods (find* and select*) which can be seen similar to the concept of stored procedures in RDMBS. Hence, DAIO may supply the query string employed by a find- or select-method and provide a WS mapping to such methods.

For EJB 2.1 CMP beans find and select methods have to be defined in the home interface:

```

264 // definition of the CMP bean's home interface
265 public interface CustomerI extends javax.ejb.EJBLocalHome {
266   ...
267   public Collection findByZipcode(String zipcode)
268     throws javax.ejb.FinderException;
269   ...
270 }

```

The actual query is defined in the deployment descriptor:

```

273 <?xml version="1.0"?>
274 ...
275 <entity>
276   <ejb-name>Customer</ejb-name>
277   <abstract-schema-name>CustomerSchema</abstract-schema-name>
278   ...
279   <query>
280     <query-method>
281       <method-name>findByZipcode</method-name>

```



```
282     <method-params>
283         <method-param>java.lang.String</method-param>
284     </method-params>
285 </query-method>
286 <ejb-ql>
287     select object(o) from CustomerSchema o where o.zipcode=?1
288 </ejb-ql>
289 </query>
290 </entity>
291
```

292 Due to the fact that CMP EJB (v. 2.1) queries are static in the above sense, DAIO has to provide
293 a service which publishes a list of services which map to the home interface methods together
294 with their description.

295 [TODO: EJB3 example]

296 6.1.3 Hibernate-QL

297 [This subsection or parts of it may go to an annex]

298 Hibernate (as well as JDO-QL and many other object query languages) supports dynamic
299 queries. They can be completely build on the client which gives the application developer the
300 same flexibility as in the case of SQL.

301 Hibernate-QL expressions are created accessing the session object:

```
302 // create the query
303 String qlExpression = "from Customer c where c.zipcode like:zipcode";
304 Query query         = session.createQuery(qlExpression);
305 query.setString("zipcode", "871%");
306
307 // query the object store
308 Collection col = query.list();
309
```

310 6.1.4 JDO-QL

311 [This subsection or parts of it may go to an annex]

312 The main difference between JDO (1.0.1) and the other described persistence technologies is that
313 JDO queries result always in the instantiation of objects in order to be able to access their
314 properties. Hibernate and EJB, e.g., allow the direct access of attributes via queries (projections).
315 JDO 2.0 will support projections, too.

316 A JDO-QL expression is created accessing the persistence manager object:

```
317 Query query = pm.newQuery(Customer.class,
318                             "zipcode.startsWith(aZipcode)");
319 query.declareParameters("String aZipcode");
320
321 Collection col = (Collection)query.executeQuery("871");
322
```

323 6.1.5 XQuery

324 [This subsection or parts of it may go to an annex]

325 XQuery is a little different from the other approaches. Almost none of the widespread ODMBS
326 and O/R mappers currently supports XQuery, yet, to the knowledge of the authors. Nevertheless,
327 it is straight forward to map XQuery queries to Hibernate-QL and JDO-QL in an implementation of
328 DAIO. So XQuery provides a query language which is completely independent of the underlying

implementation. The fact that it adheres to the XML syntax make it straight forward to use in the concepts of DAIO.

Assuming the document customers.xml contains:

```
<?xml version="1.0" encoding="utf-8"?>
<customers>
  ...
  <customer
    ...
    zipcode="87111"
  />
  ...
</customers>
```

The following query would select exactly the same as the previous queries formulated in the other query languages.

```
<collection>
{
  let $customers:=doc("input/customers.xml")/customers
  for $customer in $customers/customer
  where substr($customer[@zipcode],0,3)='871'
  return
  <customers>
    {$customer}
  </customers>
}
</collection>
```

Note that the input document does not necessarily have to exist that way. It is to highlight what the example query would do with such a document which in that case would be the mapping of the persistent attributes of the customer object to XML. The implementor of DAIO XQuery would have to take care that the XQuery query would be executed in the right way.

Furthermore, XQuery directly support static and dynamic queries in the notion of this subsection.

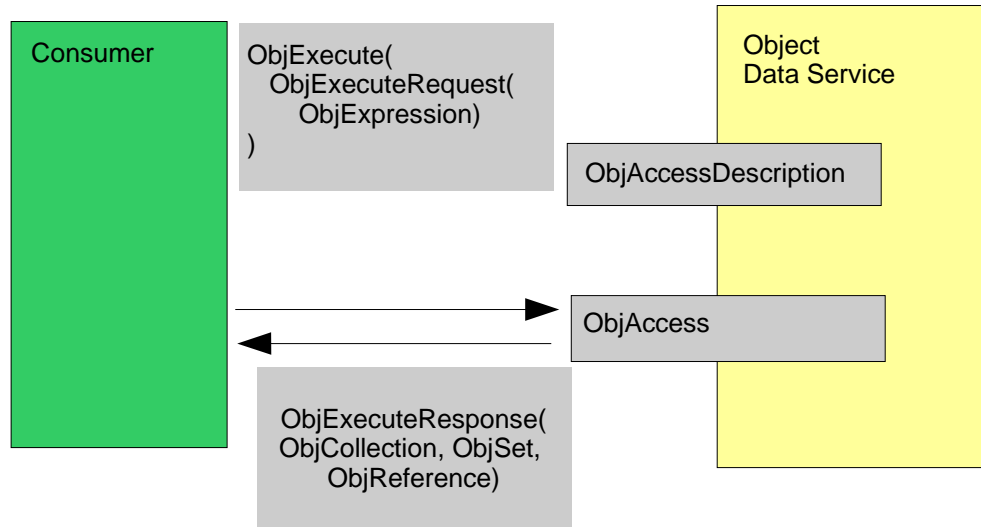
6.2 ObjAccess

ObjAccess provides access to the underlying object resources by the means of an appropriate object query language.

It is important to note that it is not the intention of DAIO to provide a solution for Object WS. How references to objects and collections or sets thereof are handled and what flavor of services is actually referenced there is out of the scope of this document (at least right now, see issues).

6.2.1 Overview

Data Access collects together messages that directly access and modify the data represented by a Data Service along with the behavioral properties that describe the behavior of these access messages, as, for example, illustrated in



371
372
373

A object storage Data Service implements the *ObjAccess* operations and exposes the *ObjAccessDescription* informational properties. In this example a consumer uses the *ObjExecute* message to submit a *ObjExpression*. The associated *ObjExecuteResponse* message will contain some combination of *ObjExecuteResponseTypeList*. The actual combination will depend upon the actual *ObjExpression*, for example:

379
380

[

381 EJB, JDO, and Hibernate queries return all Collections, Sets or Object-References which makes
382 the result representation independent of the query (language).

383 So the problem now is, are we using references to the objects? If yes, how would they look like?

384 Leon: Most object databases will require a database, federated database or server name, plus
385 an Object Identifier (OID). The wrapper could also identify the ODBMS too, in case a request
386 ends up in the wrong place.

387 Norman: Representation of references to objects. One might expect this to be some form of
388 wrapper type that states something about the source of the reference. The spec would then need
389 to state some properties of instances of the wrapper type. One might expect that the main or
390 perhaps only guarantee is that the reference can be passed as a parameter to some operations
391 on the service whence it came, where if the object still exists and appropriate permissions exist,
392 further behaviour can take place.

393

394 Page 9: Line 267: Result format: This seems never to contain the values of properties of the
395 objects, in the way that *WebRowSet* contains the values of the tuples. There needs to be some
396 representation for the values of the objects too - are there existing candidates? I note that a
397 message could contain a parameter that states how deep a copy of the properties of an object

should be returned.

]

6.2.2 Stateful Results

Queries to object resources which return collections and sets of objects contain stateful data. A object resource will instantiate those objects which can then be references. Changes to persistent attributes of such an object change its state and have to find the way into the underlying storage.

6.2.3 Stateless Results

Queries to object resources do not always return collections or sets of objects. They can in certain cases return collections or sets of attributes of objects. In the contrary to DAIR where also a modification of such a “result set” usually results in a modification of the underlying RDBMS, a modification of DAIO collections and sets does not. When dealing with object resources such results are copies and not references of the original object attributes. Hence a modification cannot result in a modification of the originally underlying object(s). Those results are stateless and easy handle in a WS context.

The following example tries to sketch the above:

The EJB-QL statement

```
select a.zipcode from CustomerSchema as a where a.zipcode=?1
```

produces a collection of attribute zipcode. The collection will contain instances of the attribute zipcode of the Customer Entity bean from page 6.1.2 8. Since the type of the attribute zipcode (java.lang.String in that particular example) is not a reference to another persistent object, the collection of zipcodes contains copies of the attribute zipcode and is hence stateless.

6.2.4 Operations

Input

Output

Select

Update

The update operation allows to change the state of one or more objects.

Delete

The update operation allows to change the state of one or more objects.

Flush

6.3**7 ObjResponse**

The result of a query is the ObjResponse. This response is composed of references to objects matching the original request. Those references are grouped in ObjCollection or ObjSet elements or return directly an ObjReference.

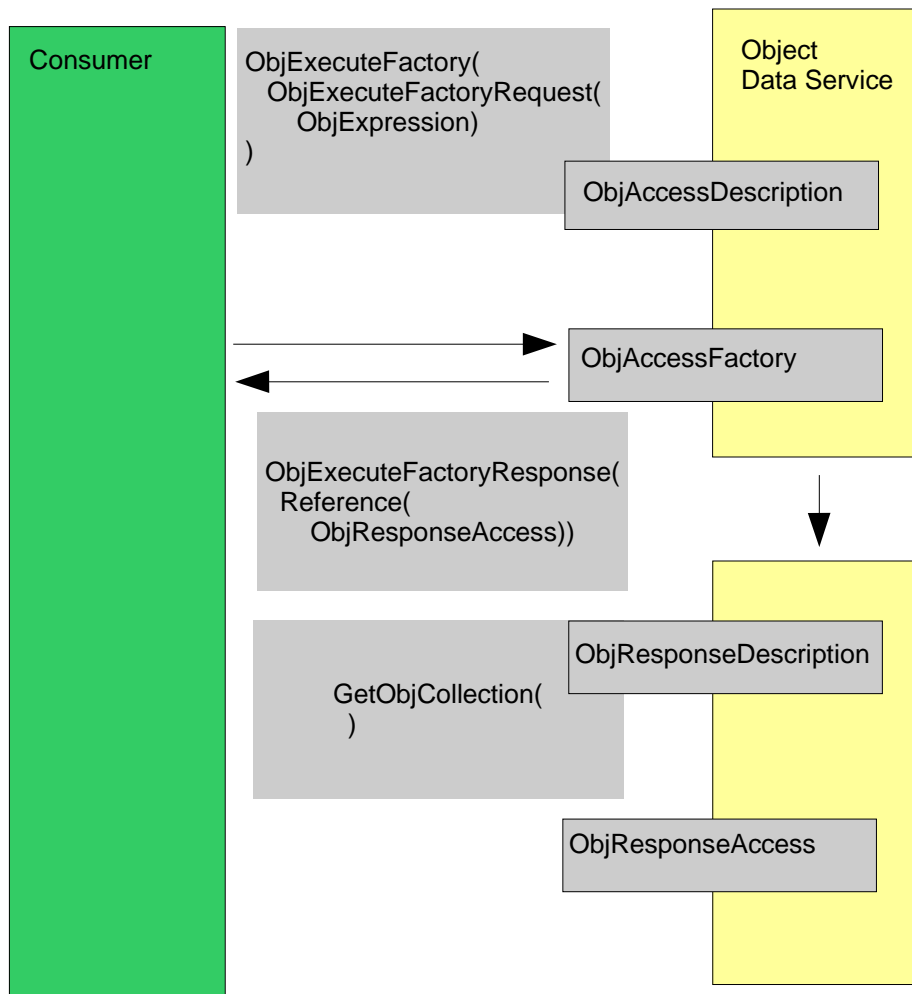
```
<xsd:element name="ObjReference">
  <!-- to be determined -->
</xsd:element>

<xsd:element name="ObjCollection">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsdaio:ObjReference" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="sid" type="xsd:id"/>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

<!-- is there a way to exclude equal elements from the sequence? -->
<xsd:element name="ObjSet">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wsdaio:ObjReference" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:any namespace="##other" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="sid" type="xsd:id"/>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ObjResponse">
  <xsd:union>
    <xsd:element ref="wsdaio:ObjCollection"/>
    <xsd:element ref="wsdaio:ObjSet"/>
    <xsd:element ref="wsdaio:ObjReference"/>
  </xsd:union>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:element>
```

8 ObjAccessFactory



486
487
488
489

9 Issues

The following lists issues which have to be dealt with.

1. How is an object to be referenced?
2. Does DAIO need to represent the object or are references enough (and the implementor has to take care of the mapping ...)?
3. If DAIO only references objects, DAIO will “just” provide means of discovering objects and their description.
4. If DAIO allows direct access of objects, e.g. by means of mapping to XML, it has to provide means of modifying those, of course. Then we are facing the problem that we need are dealing with stateful resources (the objects).
5. DAIO needs a service which provides a model description describing the returned object representations. This could happen in form of a WSDL document. Although WSDL is WS centric, we are dealing with WSs in first place. And it is possible to do wsdl2java, wsdl2idl or whatever... WSDL seems to work for now.
6. How do we deal with requests, where only some of the attributes of an object were queried? e.g. the EJB-QL statement

```
select a.zipcode from CustomerSchema as a where a.zipcode=?1
```

produces a collection of attribute zipcode. Here a kind of resultset is produced, which is not modifiable (or at least a modification does not find its way back into the original place (attribute) of the object, since the zipcode attributes are copies rather than references to the original persistent object.

7.

5 Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

6 Full Copyright Notice

Copyright (C) Global Grid Forum (2005). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."