Ann Chervenak, USC Information Sciences Institute
Karl Czajkowski, USC Information Sciences Institute

GGF DOCUMENT SUBMISSION CHECKLIST (include as front page of submission)	
	COMPLETED (X) - Date
1. Author name(s), institution(s), and contact information	(X) - June 4, 2003
2. Date (original and, where applicable, latest revision date)	(X) - June 4, 2003
3. Title, table of contents, clearly numbered sections	(X) - June 4, 2003
4. Security Considerations section	(X) - June 4, 2003
5. GGF Copyright statement inserted (See below)	(X) - June 4, 2003
6. GGF Intellectual Property statement inserted.	(X) - June 4, 2003
7. Document format	(X) - June 4, 2003

Ann Chervenak, USC Information Sciences Institute
Karl Czajkowski, USC Information Sciences Institute

Local Replica Catalog Service Specification (v2)

Status of This Memo

This memo provides information to the Grid community regarding standards for Replica Location Services. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document provides a draft grid service specification for a Local Replica Catalog, which is one component of a Replica Location Service. This specification is being developed within the OGSA Data Replication Services (OREP) Working Group of the Global Grid Forum (GGF). This specification will comply with the Grid Services Specification being developed by the Open Grid Services Infrastructure (OGSI) Working Group of the GGF.

The Local Replica Catalog Service maintains local mappings from logical names to target names and answers queries about those mappings. In this specification, we describe the Service Data Elements and Port Types that make up the Local Replica Catalog Service.

We also indicate open questions that need to be resolved by the OREP working group and indicate the parts of the grid service specification that are likely to change as the RLS evolves to a fully service-oriented architecture.

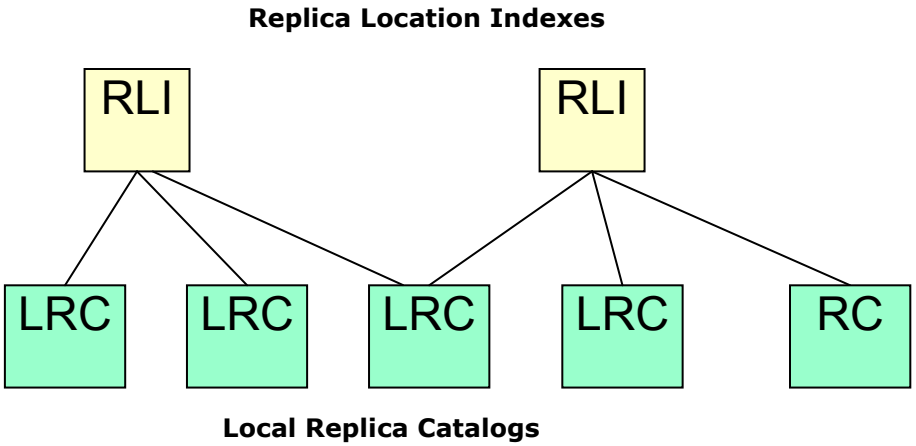
Contents

1. Introduction

In this document, we present an initial draft of a specification for a Local Replica Catalog Service. This specification is being developed within the OGSA Data Replication Services (OREP) Working Group of the Global Grid Forum (GGF). This specification will comply with the Grid Services Specification being developed by the Open Grid Services Infrastructure (OGSI) Working Group of the GGF.

We expect the LRC draft specification to change substantially from this initial draft. In particular, we expect that substantial modifications will be necessary to accommodate emerging concepts in the Open Grid Services Architecture (OGSA) relating to the treatment of datasets as first-class OGSI-compliant services. A companion document titled "Evolution of the Replica Location Service Specification to Represent Datasets as Grid Services" [3] presents a series of options or steps by which the Replica Location Service may evolve to accommodate a more service-oriented view of datasets, groups of equivalent replicas and indexes. The specification presented in this document corresponds to Option 1 of [3], which simply creates Grid service wrappers around the existing Replica Location Service but does not otherwise take advantage of the representation of datasets as services. Throughout this document, we will make note of changes that may be necessary in the specification to accommodate a more fully service-oriented Replica Location Service.

Replica Location Services maintain and provide information about data location. The initial draft of the Local Replica Catalog (LRC) Service specification is based on the following overall architecture for providing Replica Location Services (RLS) [1]. The Local Replica Catalog Service is one RLS component that maintains local mappings from logical names to target names and answers queries about those mappings. A distributed RLS may also include one or more Replica Location Index (RLI) Services, which collect mapping information from one or more LRC services and answer queries about those mappings. The RLI service is not described in this specification. An RLS may consist of a number of LRC and RLI services, where each RLI service subscribes to soft state updates from one or more LRC services. The figure below shows one possible layout of LRCs and RLIs in a distributed RLS.



Local replica catalogs (LRCs) maintain mappings between logical names (LNs) for data items, specified as URIs, and one or more other target names for these data items (either logical or physical). In many cases, the LRC will provide a mapping from an LN to one or more physical locations (URLs) of instances of the data item on storage systems; such physical names can be used by the client of the LRC to access the data item directly. In other cases, the LRC may

provide a mapping from the LN to another layer of logical naming (a URN) for the data item. The LRC can thus support multiple levels of logical naming.

One example where logical rather than physical target names would be useful is a storage management system that frequently moves data items for load balancing or archiving purposes; by registering logical names rather than physical locations as the targets of LRC mappings, the storage management system need not update LRC mappings when it moves data items. Such a storage management system would do its own internal mappings from logical target names to physical storage locations.

[Note: In a more service-oriented architecture [3], the LRC may evolve to contain mappings between logical names and Grid Service Handles (GSHs) of dataset services. A further evolution would have sets of equivalent replicas implemented as replicaSet services and replace the logical names in LRC mappings with the GSHs of the replicaSet services.]

2. Membership Management

A distributed Replica Location Service consists of a number of LRC and RLI services, each of which are considered *members* of the RLS. The RLS may respond to changes in membership, for example, when LRCs and RLIs join the distributed RLS or when they fail. For example, an RLS might react to membership changes by repartitioning the LRC services among the available set of RLI services, changing the set of LRCs to which each RLI subscribes.

There are several membership management issues that relate to the LRC service. The LRC service specification must provide a mechanism for initiating and configuring subscriptions to LRC soft state updates. The LRC service must also deal with changes to subscription characteristics and with subscriber failures.

For the first version of this specification, we will take the simplest approach to membership management. The LRC service will implement the NotificationSource PortType, which is part of the standard Grid Services Specification. Another service (or an administrator) may use the subscribe method provided by this PortType to subscribe itself (or another service) to LRC soft state service data elements. A subscription request causes the creation of a subscription service instance, which can then be used by clients to manage the lifetime of the subscription. The LRC service continues sending soft state updates to the subscriber until the subscription lifetime expires.

Another approach to membership management that may eventually be adopted is to allow a service (such as an RLI) to subscribe to notifications of LRC locator information. Locator state informs the subscriber that the LRC service exists. By sending locator information to a subscriber, the LRC effectively invites that service to initiate a separate subscription to its soft state update service data elements.

[Note: In a fully service-oriented RLS architecture, RLS index services (LRCs and RLIs) could be implemented using either OGSA indexing mechanisms or OGSI ServiceGroups [2]. RLS membership could be managed by constructing an external registry service that maintains information about the collection of participating RLS index services. This registry service could include methods that react to changes in RLS membership by redistributing soft state update subscriptions among RLS index services in the hierarchy. Such a registry service could be implemented using OGSI ServiceGroups.]

3. Subscribing to LRC Soft State Updates

LRC services currently provide two types of soft state updates. The first type consists of a list of logical names that are registered in the catalog. The second type of soft state update is a summary of the contents of the LRC called a Bloom Filter Summary, which is constructed by performing a series of hash functions on each logical name registered in the LRC. For each hashed value, the corresponding bits of the bloom filter are set.

As already mentioned, the LRC service allows clients to subscribe to soft state service data elements by implementing the standard NotificationSource PortType that is described in the Grid Services Specification. Service data elements of the subscription will indicate whether the notification will use bloomFilterCompression and the frequency of soft state updates.

3.1 NotificationSource PortType: Service Data Descriptions

The Service Data Descriptions below describe the two types of soft state updates to which a client may subscribe for notification of state changes: a list of logical names registered in the LRC or a bloom filter summary of LRC state.

```
<gsdl:serviceDataDescription
        name="logicalNames"
        type="xsd:anyURI"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="mutable">
   <wsdl:documentation>
        logicalNames is the list of logical names registered in the LRC. Each logical name is a
        URI.
   </wsdl:documentation>
</gsdl:serviceDataDescription

<gsdl:serviceDataDescription
        name="bloomFilterSummary"
        type="xsd:string"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="mutable">
   <wsdl:documentation>
        The bloom filter summary of the contents of the LRC is implemented by this Grid Service.
        The summary is a bit array, which is currently defined to be a string.
   </wsdl:documentation>
</gsdl:serviceDataDescription
```

[One open design issue is whether these Service Data Descriptions should be separately defined (as above), or whether we should take a more polymorphic approach, using a single Service Data Description to represent LRC state and differentiating the type of soft state update via different query mechanisms.]

3.2 The NotificationSource PortType: Operations

Operations supported by the standard NotificationSource PortType include Subscribe and SubscribeByServiceDataName.

The subscription operation should specify the type of soft state subscription, which may be one of the following:

- *LogicalNames*

- *bloomFilterSummary*

[One issue that does not seem to have adequate support in the current Grid Services Specification is the need for sending partial service data to a subscriber. We are interested in two types of partial service data updates: we would like to be able to partition the logical file namespace that is sent to a particular subscriber, for example, sending only those logical names that match a specified pattern; and we would like to send incremental updates that consist of only those service data elements that have changed since the previous update. Neither of these types of updates appears to be possible with the current GSS, which sends only complete service data updates.]

[Note: In a fully service-oriented design, we would still likely want to implement optional Bloom Filter Compression to reduce the size of soft state updates among index services in the hierarchy. If RLS index services are implemented using OGSA index mechanisms, then the service provider associated with datasets or indexes would be responsible for producing these soft state updates, including bloom filters. If the RLS index services are implemented as ServiceGroups, we would need to extend existing ServiceGroup service data and port types to expose soft state updates and allow subscription.]

4. The ServiceAdministration PortType

The ServiceAdministration PortType provides Service Data Descriptions and methods related to the configuration of the LRC server.

4.1 ServiceAdministration PortType: Service Data Descriptions

The Service Data Descriptions below describe data items that describe LRC configuration and dynamic service state that can be queried by the FindServiceData command implemented by the GridService PortType.

[One open question is how LRC configuration and state information should be aggregated for FindServiceData queries. Initially, we have grouped together some related LRC configuration information into complex data type and left others as individual service data elements.]

```
<xsd: complexType name="bloomFilterParameters">
        <xsd:sequence>
                <xsd: element name="bloomfiltermodel" type="xsd:string" />
                <xsd: element name="hashfunctiondescription" type="xsd:string" />
                <xsd: element name="bloomfilternumhash" type="xsd:int" />
                <xsd: element name="bloomfilterratio" type="xsd:int" />
        </xsd:sequence>
</xsd:complexType>

<gsdl:serviceDataDescription
        name="bloomFilterConfiguration"
        type="tns:bloomFilterParameters"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="constant">
   <wsdl:documentation>
        Parameters relating to the setup of an LRC for bloom filter compression. The
        bloomfiltermodel describes the model being used for bloom filter compression. The
        hashfunctiondescription characterizes the hash functions being used to generate the
```

bloom filters. The bloomfilternumberhash value is the number of hash functions used in creating the bloom filters; the default value is 3. The bloomfilterratio value is the ratio of bloom filter size to the approximate number of LNs in the LRC; the default value is 10. We should configure the LRC to generate multiple bloom filters with different parameters; thus there can be an unbounded number of Bloom Filter Configurations.
</wsdl:documentation>
</gsdl:serviceDataDescription

<gsdl:serviceDataDescription
name="serviceUptime"
type="xsd:int"
minOccurs="1"
maxOccurs="1"
mutability="mutable">
<wsdl:documentation>
Specifies how long the LRC service has been operational. *[Note: this service data element will likely be eliminated when the Common Resource Model port type is available as part of the OGSI standard..]*
</wsdl:documentation>
</gsdl:serviceDataDescription

<gsdl:serviceDataDescription
name="numberSubscribers"
type="xsd:int"
minOccurs="1"
maxOccurs="1"
mutability="mutable">
<wsdl:documentation>
Specifies how many services have subscribed to this service. *[Note: this service data element will likely be eliminated when the Common Resource Model port types is available as part of the OGSI standard..]*
</wsdl:documentation>
</gsdl:serviceDataDescription

<gsdl:serviceDataDescription
name="numberRegisteredLogicalNames"
type="xsd:int"
minOccurs="1"
maxOccurs="1"
mutability="mutable">
<wsdl:documentation>
Specifies how many logical names are registered in the LRC service.
</wsdl:documentation>
</gsdl:serviceDataDescription

<gsdl:serviceDataDescription
name="numberRegisteredTargetNames"
type="xsd:int"
minOccurs="1"
maxOccurs="1"
mutability="mutable">
<wsdl:documentation>
Specifies how many target names are registered in the LRC service.
</wsdl:documentation>
</gsdl:serviceDataDescription

```
<gsdl:serviceDataDescription
        name="numberRegisteredMappings"
        type="xsd:int"
        minOccurs="1"
        maxOccurs="1"
        mutability="mutable">
   <wsdl:documentation>
        Specifies how many mappings from logical to target names are registered in the LRC
service.
   </wsdl:documentation>
</gsdl:serviceDataDescription


<xsd: complexType name="definedAttributeInformation">
        <xsd:sequence>
                <xsd: element name="attributeName" type="xsd:string" />
                <xsd: element name="objectType" type="xsd:string" />
                <xsd: element name="attributeType" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>

<gsdl:serviceDataDescription
        name="definedAttributeList"
        type="tns:attributeInformation"
        minOccurs="0"
        maxOccurs="unbounded"
        mutability="mutable">
   <wsdl:documentation>
        List of user-defined attributes that have been specified for logicalNames and
        targetNames in the LRC.
   </wsdl:documentation>
</gsdl:serviceDataDescription
```

4.2 ServiceAdministration PortType: Operations and Messages

The methods currently supported in the ServiceAdministration PortType involve creation and deletion of attribute definitions.

4.2.1 ServiceAdministration :: CreateAttribute

This function defines a new attribute that will be associated with either a logicalName or a targetName.

Input:

- *AttributeName*
- *ObjectType*: specifies whether the new attribute is associated with a logicalName or targetName
- *AttributeType*: specifies the type of the attribute (either integer, floating point, string or date)

Output:
Fault(s):

4.2.2 ServiceAdministration :: DeleteAttribute

This function deletes an existing attribute that is associated with either a logicalName or a targetName.

Input:

- *AttributeName*
- *ObjectType*: specifies whether the new attribute is associated with a logicalName or targetName
- *ClearValues*: If this Boolean value is true, then any values for this attribute are first removed from the objects with which they are associated. If false, then if any values exist, an error message is returned.

Output:
Fault(s):

[Note: In a fully service-oriented architecture, the use of attributes is likely to change significantly. Attributes that are currently associated with a targetName would become service data elements of the dataset service. Attributes that are currently associated with the logicalName would become service data elements of the replicaSet service that represents the equivalence set of replicated datasets. An LRC index service might cache these attribute values for more efficient search operations or may only store references to the dataset or replicaSet services where the attributes are located.]

5. The MappingsAdministration PortType

The MappingsAdministration PortType provides Service Data Descriptions and methods associated with creating and deleting LRC mappings and attributes.

5.1 MappingsAdministration PortType: Service Data Descriptions

5.2 MappingsAdministration PortType: Operations and Messages

5.2.1 MappingsAdministration :: AddMapping

Creates a mapping in the LRC between a logical name (a URI) and a target name. The target name is specified as a URI and may be either a physical location (URL) or a logical name (URN).

Input:
- *LogicalName*: A logical name that is already registered in the catalog.
- *TargetName*: The name (a URI) to which the logical name is mapped. This name may be either a physical name (URL) or logical name (URN).

Output:
Fault(s):

5.2.2 MappingsAdministration :: DeleteMapping

Deletes an existing mapping in the LRC between a logical name and a target name.

Input:
- *LogicalName*: A logical name in an existing mapping in the catalog.
- *TargetName*: A name to which the logical name is mapped in an existing catalog mapping.

Output:
Fault(s):

5.2.3 MappingsAdministration :: AddAttributeValue

This method adds a new attribute value that will be associated with either a logicalName or a targetName.

Input:
- *Key*: LogicalName or TargetName that identifies the object to which the attribute value should be added.

- *ObjectType*: specifies whether the new attribute is associated with a logicalName or targetName
- *AttributeType*: specifies the type of the attribute (either integer, floating point, string or date)
- *AttributeName*: The name of an attribute previously created by the CreateAttribute method of the ServiceAdministration PortType.
- *AttributeValue*

Output:
Fault(s):

5.2.4 MappingsAdministration :: RemoveAttributeValue
This method removes an existing attribute value that is associated with either a logicalName or a targetName.

Input:
- *Key*: LogicalName or TargetName that identifies the object with the attribute value that will be removed.
- *ObjectType*: specifies whether the new attribute is associated with a logicalName or targetName
- *AttributeName*: The name of an attribute to be removed

Output:
Fault(s):

[Note: The working group should discuss whether there is a need for a method to add/remove multiple attributes atomically or to atomically add a mapping and a set of associated attributes. The current port type requires separate method invocations for each mapping and each attribute. While this requires more method calls, it places relatively few constraints on the implementation. By contrast, if complex, atomic operations are required, they would require the LRC to implement fairly complex transactions.]

[Note: In a fully service-oriented architecture, the methods specified in this port type may disappear. A client would not directly create or delete mappings or attribute values in the LRC. Rather, these attributes would be associated with dataset or replicaSet services. The LRC would subscribe to these services and index the service content (mappings and attributes) to provide efficient search capabilities.]

6. The QueryMappings PortType

The QueryMappings PortType provides Service Data Descriptions and methods associated with querying existing LRC mappings and their attributes.

[An important open question for the working group is whether the LRC content should be queried using the methods specified here, or whether the contents of the LRC should be exposed as grid service state and represented using service data elements. In the latter case, we would need richer query mechanisms than are currently available from Grid Services to provide all the wildcard searches, etc., that are currently available through the QueryMappings port type. In particular, we would need to provide a mechanism to map from the XPATH queries that will be supported on grid service data to the query mechanism used by the LRC service implementation, e.g., SQL.]

6.1 QueryMappings PortType: Service Data Descriptions and Elements

6.2 QueryMappings PortType: Operations and Messages

6.2.1 QueryMappings :: QueryLogicalNameExists
Checks whether there is an existing entry in the LRC for the specified logical name.

Input:
- *LogicalName*: A logical name (URI)

Output:
Fault(s):

6.2.2 QueryMappings :: QueryTargetNameExists
Checks whether there is an existing entry in the LRC for the specified target name.

Input:
- *TargetName*: This name (a URI) may refer to either a physical location or a logical name.

Output:
Fault(s):

6.2.3 QueryMappings :: GetMappingsWithLogicalName
Returns all {logicalName, targetName}mappings registered in the LRC that contain the specified logicalName.

Input:
- *LogicalName*

Output:
- *{LogicalName, TargetName} mappings*: One or more registered mappings that contain the specified logicalName.

Fault(s):

6.2.4 QueryMappings :: GetMappingsWithTargetName
Returns all {logicalName, targetName}mappings registered in the LRC that contain the specified targetName.

Input:
- *TargetName*: This name (a URI) may refer to either a physical location or a logical name.

Output:
- *{LogicalName, TargetName} mappings*: One or more registered mappings that contain the specified targetName.

Fault(s):

6.2.5 QueryMappings :: WildcardMatchLogicalNamePattern
Returns all {logicalName, targetName}mappings registered in the LRC that match the wildcard expression for logicalNamePattern.

Input:
- *LogicalNamePattern*: Pattern to be used in wildcard match against logical names registered in the LRC.
- *PatternType*: Specifies interpretation of wildcard characters.

Output:
- *{LogicalName, TargetName} mappings*: One or more registered mappings whose logicalName values match the pattern specified by logicalNamePattern.

Fault(s):

6.2.6 QueryMappings :: WildcardMatchTargetNamePattern
Returns all {logicalName, targetName}mappings registered in the LRC that match the wildcard expression for targetNamePattern.

Input:

- *TargetNamePattern*: Pattern to be used in wildcard match against target names registered in the LRC.
- *PatternType*: Specifies interpretation of wildcard characters.

Output:

- *{LogicalName, TargetName} mappings*: One or more registered mappings whose targetName values match the pattern specified by targetNamePattern.
-

Fault(s):

6.2.7 QueryMappings :: GetAttributeDefinition

Returns{attributeName, objectType, attributeType}definitions for specified attribute name and object type.

Input:

- *AttributeName*
- *ObjectType*: specifies whether the new attribute is associated with a logicalName or targetName

Output:

- *{AttributeName, ObjecctType, AttributeType}* definitions of attributes, where *AttributeType* specifies the type of the attribute (either integer, floating point, string or date)

Fault(s):

6.2.8 QueryMappings :: GetAttributeValue

Returns{attributeName, objectType, attributeType, attributeValue}tuple for specified query.

Input:

- *Key*: LogicalName or TargetName that identifies the object with which the attribute value is associated.
- *AttributeName*: The name of an attribute to retrieve. If NULL all attributes for the specified Key and ObjectType are returned.
- *ObjectType*: specifies whether the attribute is associated with a logicalName or targetName

Output:

- *{AttributeName, ObjectType, AttributeType, AttributeValue}* results, where AttributeType specifies the type of the attribute (either integer, floating point, string or date)

Fault(s):

6.2.9 QueryMappings :: SearchAttributesByValue

Returns{attributeName, objectType, attributeType, attributeValue}tuple for attributes that match the specified comparison operation.

Input:

- *AttributeName*: The name of an attribute to retrieve. If NULL all attributes for the specified Key and ObjectType are returned.
- *ObjectType*: specifies whether the attribute is associated with a logicalName or targetName
- *Operation*: specifies the logical operation performed by the search query. Possible options include:
    - *Return all values*
    - *Return values matching operand1*
    - *Return values not matching operand1*
    - *Return values greater than operand1*
    - *Return values greater than or equal to operand1*
    - *Return values greater than or equal to operand1*

  - *Return values less than or equal to operand1*
  - *Return values less than or equal to operand1*
  - *Return values between operand1 and operand2*
  - *Return strings "like" operand1 (SQL like)*
- *Operand1Type*: may be integer, floating point, string or date
- *Operand1Value*
- *Operand2Type*: (optional) may be integer, floating point, string or date
- *Operand2Value*: (optional)

Output:
- *{AttributeName, ObjectType, AttributeType, AttributeValue}* results, where AttributeType specifies the type of the attribute (either integer, floating point, string or date)

Fault(s):

[An open issue is how to support richer queries across multiple attributes. The SearchAttributesByValue method specified here provides a limited set of operations on one or two operands. A more general query mode is requiredl. If the state of the LRC is exposed as service data, then the query model corresponds to that available for service data in the GridService PortType.]

[Note: In a fully service-oriented architecture, some of these methods may change or be eliminated, depending on whether the LRC indexes attributes.]

7. The GridService PortType

As specified by the Grid Services Specification, the LRC must implement the GridService PortType. The GridService portType encapsulates behavior including querying against the serviceDataSet of the Grid service instance and managing the termination of the instance.

Operations provided by the GridService PortType include FindServiceData, queryByServiceDataName, setTerminationTime for the service, and Destroy. In particular, the FindServiceData method can be used to query any of the service data elements whose descriptions are defined in this document. In particular, the FindServiceData method can be used to synchronously query the soft state of the LRC, effectively forcing a synchronous soft state update.

8. Security Considerations

Our proposal is for an OGSI-Compliant grid service. Therefore, our service will have all the same security capabilities and issues as other OGSI-compliant services. Additional security considerations relating to this specification are described in [3].

Author Information

Ann L. Chervenak, USC Information Sciences Institute, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA, annc@isi.edu

Karl Cjakowski, USC Information Sciences Institute, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA, karlcz@isi.edu

Intellectual Property Statement

Full Copyright Notice

References

[1] “Giggle: A Framework for Constructing Scalable Replica Location Services”, Ann Chervenak, Ewa Deelman, Ian Foster, Leanne Guy, Wolfgang Hoschek, Adriana Iamnitchi, Carl Kesselman, Peter Kunszt, Matei Ripenu, Bob Schwartzkopf, Heinz Stocking, Kurt Stockinger, Brian Tierney to appear in Proceedings of SC2002 Conference, November 2002.

[2] “Open Grid Services Infrastructuure (OGSI)”, S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, P. Vanderbilt, Global Grid Forum Open Grid Services Infrastructure (OGSI) Working Group, February 17, 2003.

[3] “Evolution of the Replica Location Service Specification to Represent Datasets as Grid Services”, Ann Chervenak and Karl Czajkowski, Global Grid Forum OGSA Data Replication Services (OREP) Working Group, June 4, 2003.