GWD-I (Draft)
OGSA Data Working Group

D. Berry, NeSC
A. Luniewski, IBM
M. Antonioletti, EPCC

September 7, 2007

OGSA™ Data Architecture

Status of this Document

This document provides information to the community regarding the specification of the data architecture of the Open Grid Services Architecture (OGSA). It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Trademarks

Abstract

The Open Grid Services Architecture (OGSA) presents a vision of a broadly applicable and adopted framework for distributed system integration, virtualization, and management. OGSA provides several capabilities, one of which is data management. This document, produced by the OGSA Data Working Group within the Open Grid Forum (OGF), gives a high-level description of the interfaces, behaviors, resource models, and bindings for manipulating data within the broader OGSA architecture. The functionality described covers the storage, movement, access, replication, caching and federation of files and databases.

Contents

1 Introduction

This document is a product of the OGSA Data Architecture Working Group (OGSA-D WG) of the Open Grid Forum (OGF). It describes and guides the design of the data management capability of the Open Grid Services Architecture (OGSA); it addresses the storage, movement, access and manipulation of data within an OGSA framework. The OGSA-Data WG has worked in conjunction with the OGSA WG, which guides the overall development of OGSA.

This informational document is one component of a set of documents that, over time, will fully define OGSA, both informatively and normatively. For an overview of the OGSA architecture, see the Open Grid Service Architecture version 1.5 [OGSA]. The full document set, the status, and the planned evolution of OGSA are described in *Defining the Grid: A Roadmap for OGSA Standards* **Error! Reference source not found.** Of particular relevance is a companion to the current document that will describe data architecture scenarios [Scenarios].

The authors intend that this architecture will incorporate and integrate relevant insights and documents from other groups, especially other OGF Working Groups. For example, it already incorporates work from the Grid File System Working Group, the Database Access and Integration Working Group, the ByteIO Working Group and the Grid Storage Management Working Group. However, the emphasis has been on producing a coherent architecture, so we do not guarantee that everything produced by these other groups has been adopted.

1.1 Terminology

The OGSA Data Working Group is comprised of members from many different organizations and communities within which it is common for a term to have many different interpretations. Even the term “data” is interpreted differently or at least given different emphases, as discussed in the “Scope” section below. Thus, the OGSA Data WG has taken great care to define the terms that it uses. In this document, as in other OGSA documents, italicized terms are defined in a glossary, see Section 15. For particularly common terms, such as “service” or “interface”, only the first few occurrences are in italics. The relevant entries are given in a glossary section at the end of this document. Many of these entries also appear in the main OGSA Glossary document [OGSA Glossary].

The authors have noted that the term metadata is one that causes particular confusion in discussions about data. So, to avoid any confusion about the term, we note that its primary meaning is “data about data”, which can include descriptions of data structure, provenance information (i.e. where the data came from and how it was obtained) and third-party annotations. In the context of service-oriented architectures, the term is also used to describe properties of services or of the resources that they provide.

In this document, we attempt to minimize the use of the term “metadata”. Instead, we prefer to use more precise terms such as “data description”, “service description” or “resource description”.

1.2 Services and interfaces

In OGSA terminology, *services* are software components, participating in a service-oriented architecture, that provide functionality and/or contribute to realizing one or more *capabilities*. The operations that a service offers are specified by its *interface*. The OGSA architecture defines common *interfaces* for similar types of *service*, thereby hiding differences in their properties and operations, allowing them to be viewed and/or manipulated in a common way. These common interfaces *virtualize* the *resources* provided by the *services*. Figure 1 presents a schematic overview of the inter-relationship between these terms.

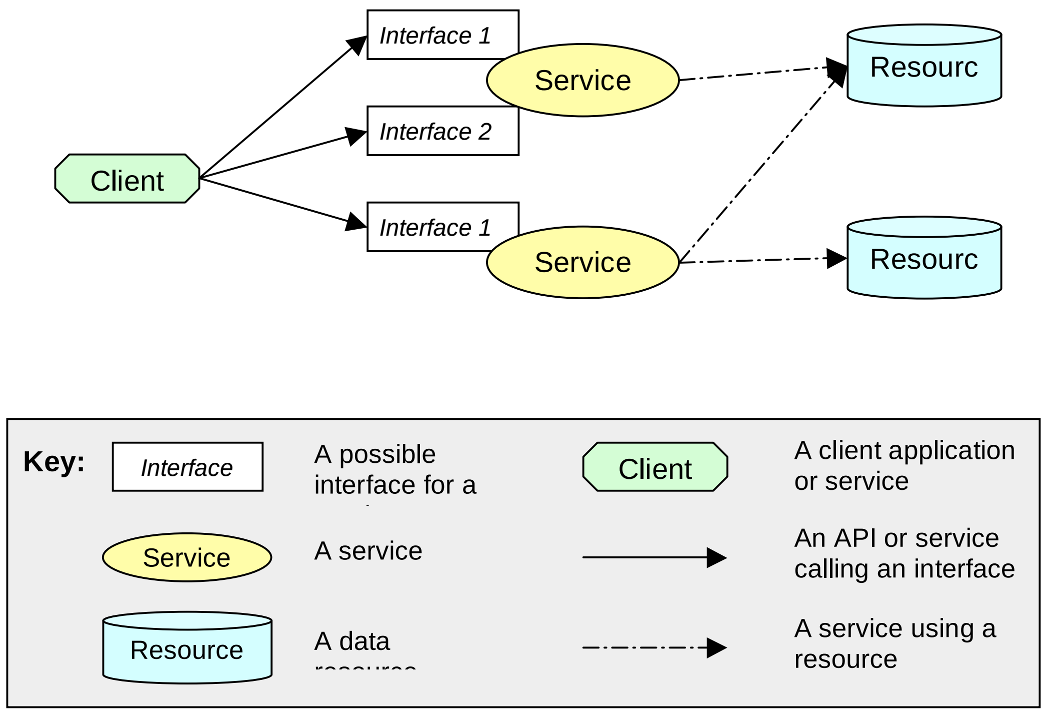
Interface 1
Service
Resourc
Interface 2
Client
Interface 1
Service
Resourc
Key:
Interface
A possible interface for a
Client
A client application or service
Service
A service
An API or service calling an interface
Resource
A data
A service using a resource

Figure 1: Services, interfaces and resources

This document primarily describes the *interfaces* that *virtualize* the *services* managing the transfer, storage, access and federation of data resources. It also describes *interfaces* for *data location management* services such as the staging, caching and replication of data. It discusses some of the underlying relationships that must be maintained by services implementing OGSA data capabilities. It also demonstrates how appropriate combinations of data services can realize these capabilities.

Virtualization also allows data functionality to be provided by computational *resources* or other types of *resource*. For example, a *service* might implement a query by calculating a result value on the fly - it is not relevant to the *data access* virtualization whether the data is generated on the fly or whether it is materialized and stored (although this may be revealed by a corresponding *management* interface or properties that describe the currency of that data).

If a data service is associated with more than one data resource, it may be necessary to disambiguate the data resource at which messages are targeted. The OGSA naming scheme can be used to name each resource provided by a service. Many existing data resources, such as database management systems, already have established names and lifetimes both of which may not be controlled by the service layer, for example the names of databases and tables in relational database management systems. This may require alternative means for exposing the names of data held in these resources.

1.3 Scope

The OGSA Data architecture presents a “toolkit” of *data services* and *interfaces* that can be composed in a variety of ways to address multiple *scenarios*. These services and interfaces include *data access*, *data transfer*, *storage management*, *data replication*, *data caching*, and *data federation*.

The components of the data architecture can be put together to build a wide variety of solutions. A companion document on data architecture *scenarios* [Scenarios] will describe a number of such uses. This set of scenarios is intended to be exemplary, not all inclusive. However, it is a goal of the current document to describe the data architecture in sufficient detail and specificity that interoperable data architecture implementations are the norm, rather than the exception.

When reading this document, it is important to note that the term “data” might fall under many categories including the following:

- A sequence of bytes, without name or interpretation of structure;
- Files or sets of files, without interpretation of their contents;
- Structured data, which may be contained within files, streams or database management systems.

The scope of topics needed for a complete data architecture is enormous. To cover the entire scope of data would take far longer than is reasonable. Thus, this initial version of the document and the architecture that it describes is limited in its scope. The primary focus of this version of the architecture is for data that lives in file systems and data that lives in database systems. The document does not discuss queries and transformations over data streams – a very important data source but one whose definition and incorporation is too major a topic to be attacked in this initial version. We also do not address sensor data, a very important sub case of data streams.

Similarly, this initial version of the document does not describe all the data-oriented services that may be required to build more advanced data Grid functionality. The omissions include common systems such as distributed file systems, “metadata” management systems, and distributed personal space management systems. .

We expect this document to evolve over time, with future versions extending the scope of topics covered and the range of services described. For a further vision of how data services could form part of a comprehensive Grid solution, from a database-centric viewpoint, see **Error! Reference source not found.**.

1.4 Document structure

The rest of the document is structured as follows. Section 2 gives a brief overview of the various aspects of the data architecture. Section 3 provides background for readers unfamiliar with the OGSA architecture in general, by describing the non-data parts of the OGSA architecture that are built on, referred to or used by the capabilities presented in this document. Readers familiar with the OGSA architecture should feel free to skip this section.

Section 4 discusses security issues, highlighting those that particularly apply to data, and section 5 describes operations for setting policies. Section 6 discusses how data is described within the architecture. This includes the description of data formats and data resources by the properties of a data service. It also includes third-party descriptions that may be stored in a registry.

Sections 7 to 13 each describe in detail one of the capabilities that are realized by this data architecture. For each of these capabilities, the relevant section starts with a summary table of the operations provided by this capability and then explains how the capability is realized and gives a detailed description of the relevant services, operations and virtualizations.

The description of capabilities begins with the transfer of data between services or resources, move on to the accessing of data in resources, and then follow with the management of storage resources. Further sections discuss caching, data replication and federation services, and data catalogs.

Section 14 is a brief conclusion. Appendix 15 summarizes the interfaces described in this document. Appendix 16 maps some of these interfaces to items in existing specifications. Appendix 17 gives a guide to the specifications referenced in this document. Finally, the glossary gives definitions of the terms used in this document.

2 Overview

In this section we briefly discuss key aspects of the data architecture to give the reader an overview. In many cases these aspects are discussed in greater detail in sections 4 to 12. In some cases, however, the aspect is discussed only here.

2.1 Levels of abstraction

A distributed system may contain a variety of data resources. These resources may use different data models to structure the data, different physical media to store it, different software systems to manage it, different schema to describe it, and different protocols and interfaces to access it. The data may be stored locally or remotely; may be unique or replicated; may be materialized or derived on demand. OGSA data services can provide different levels of *virtualizations* over these data resources. Virtualizations provide abstract views that hide these distinctions and allow the data resources to be manipulated without regard to their nature.

Conversely, although the data services allow the clients to ignore these distinctions, some clients may prefer to exploit them. For example, a client may wish to make use of a particular query language for a given database, or to specify the location of a particular data resource to use. One client may require native access to the data, while another needs to tune the performance parameters of the data resource. To support such clients, a data service may provide multiple interfaces or offer several options for query languages, transfer protocols, or such like. These layered interfaces allow clients to choose the combination of performance and abstraction that suits them best.

We have found that different communities apply different emphases on the capabilities and levels of abstraction that Grid data services must provide for the data they expose to Grids. Some examples will be given in the companion scenarios document [Scenarios]. For the OGSA architecture to be truly general, it must be able to operate under all of these example scenarios and more. Fortunately, there are several commonalities among the required functionalities that allow us to propose an integrated architecture. These are outlined in the remainder of this section.

2.2 Client Libraries

While access to data resources is possible using the service interfaces directly, we also expect that OGSA data services will be accessed via APIs. Libraries for use by client applications could map these APIs to the corresponding messages in the Service Oriented Architecture (SOA) framework. Often these libraries will implement existing, legacy, APIs. This will allow easy integration of OGSA data services with existing applications for backwards compatibility.

The key idea here is a front end to the data supported by an implementation that speaks to the service-oriented architecture. As examples, a Grid-aware NFS V3 library [NFS] will look like a standard NFS mount point, a caching service could be used by NFS or CIFS libraries [CIFS], a POSIX interface [POSIX] could make remote files seem local, and a JDBC [JDBC] or ODBC [ODBC] interface will make Grid database resources appear local.

Other client libraries may be specifically written with Grids in mind. An example is Globus XIO [XIO], which provides a portable API to swappable IO implementations. This advanced interface can handle asynchronous operations in a threaded environment.

Although the architecture is designed to meet the needs of multiple client libraries, the definition of the library services that map from APIs to service operations falls outside the scope of this document.

2.3 Virtualized Resources

The basic entities of this data architecture are illustrated in **Error! Reference source not found.**. *Resources* are managed by *data services* that may:

- provide *data access* interfaces,
- provide *data source* or *data sink* interfaces for *data transfer* operations, and

- describe data via properties (*data description*).

Some *resources*, such as file systems and databases, are storage based and this architecture also includes interfaces for *storage management*.

Data access provides a means of inputting or outputting data via a service to its client. A client can specify the data that is of interest. For example, a client might specify the subset of a file that is of interest or provide a SQL query to select the data that is of interest. Data might be returned in the response message to a data access operation if the amount of returned data is small. Alternatively, a data transfer service might be used to move large amounts of data using mechanisms such as GridFTP or storage level data movement facilities.

Thus *Data transfer* provides an explicit and controllable means by which data may be moved from a *data source* to a *data sink* (i.e. a consumer). *Data transfer* services control the transfer of data between *resources* and/or *consumers* and provide a transfer control *interface*.

All these *interfaces* are used by clients, which may be other *services*, or *APIs* such as those contained in SAGA, POSIX and NFS specifications.

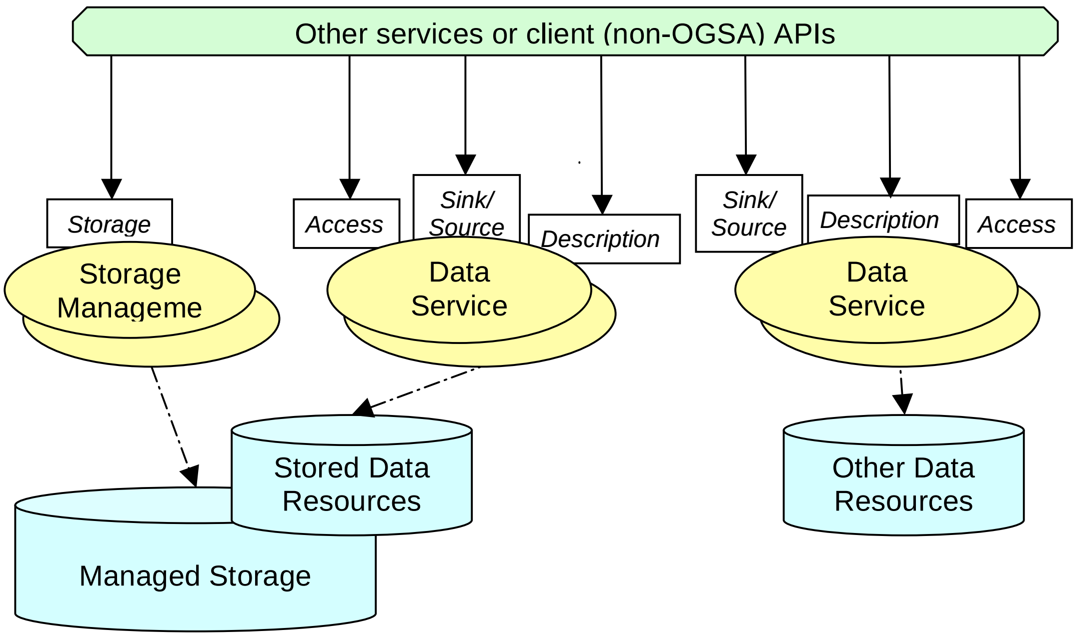

Figure 2: Basic entities in the data architecture

This diagram presents a schematic and abstract overview. In reality, some services may only support an access interface or a transfer interface. Some operations may combine functionality from several interfaces, e.g. some access operations might specify that the results should be transferred to another resource; for particularly simple transfers, operations might not require a separate data transfer service.

Generic management and provisioning interfaces have been omitted, as have security services and their related permission decision points.

2.4 Usage Patterns

Services using access interfaces generally fall under one of the following interaction patterns:

- Request-response: a client sends an access request and any data generated is included in the response sent back to the consumer. Alternatively, the request may contain data which is to be added to the resource and the response will return the status of this operation back to the consumer.

- Third party delivery: a data request includes instructions for delivering the data to a third party via some data transfer mechanism, such as the Data Transfer mechanism described in this document. This allows large amounts of data to be transferred efficiently through some suitable protocol.
- Factory: a request sent by a consumer produces a new resource that is populated by the results obtained from a request. This is created by the service and is maintained at the service end. The data in this new resource may be collected at some later stage by the same consumer or a third party. This forms an indirect form of third party delivery.
- Bulk load: data is uploaded to a data resource through a service by a consumer or third party, using a data transfer mechanism.

In this version of the architecture we predominantly concentrate on the first three patterns.

2.5 Moving data: Transfer and Replication

The data architecture has to be able to handle the transfer of large amounts of data to and from resources. Large-scale data can be transferred using dedicated protocols, as using SOAP messages would be far too slow. Data transfer implementations must transfer data in as efficient a manner as is practical.

Figure 3**Error! Reference source not found.** shows how data transfer fits within the data architecture. One use of data transfer is to return the result of a data access operation when the result is (or is expected to be) too large for SOAP to handle efficiently. This is indicated in the diagram below by the data transfer arrow between the resource and the client.

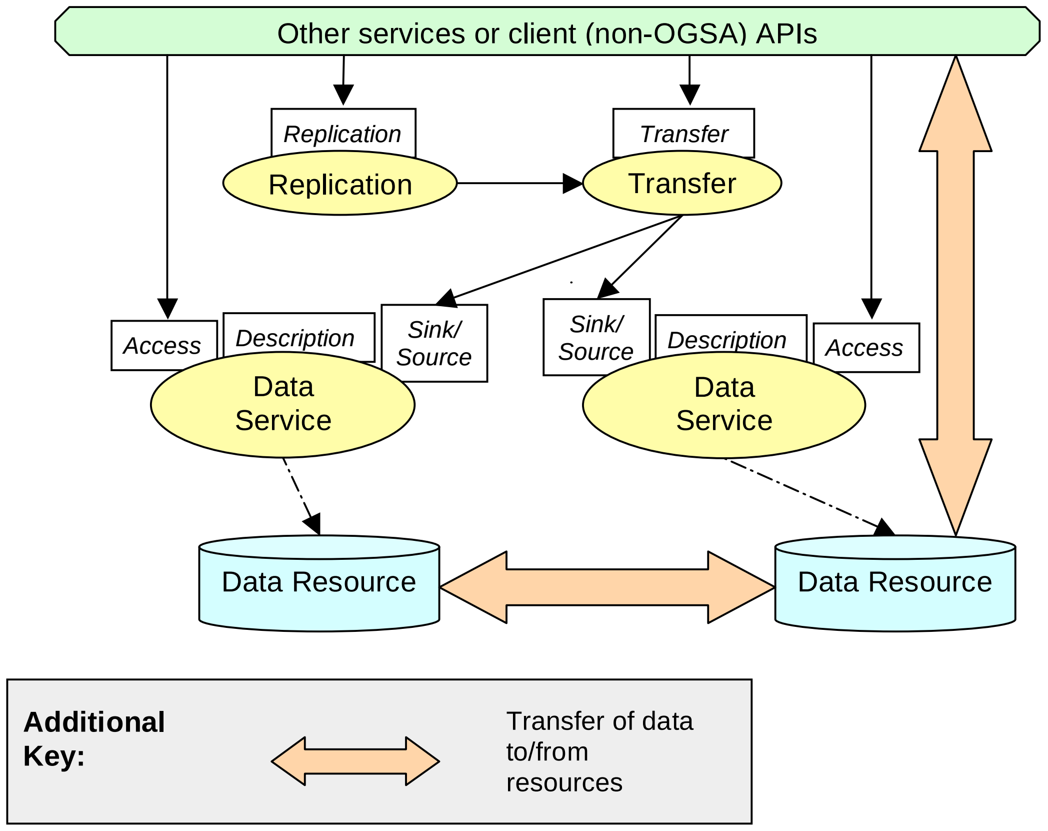

Figure 3: Transfer and replication in the data architecture

Dedicated transfer services can be used to control the transfer of data between resources. These call operations on the *data source* and *data sink* interfaces to set up and initiate the transfer. Transfer operations are not restricted to linking resources that are managed by data services.

Data that is encoded in a defined syntax is called a *data set.* For example it might be used for externalization from a resource, ready for transfer. So, for example, a *data access* operation could select data which is then encoded as a *data set*, and then a *data transfer* mechanism is invoked to manage the transfer from the *resource* to the *client* or another *consumer*. Actual transfer of data uses an appropriate protocol and takes as direct a path as possible.

Data replication services can use transfer services to maintain replicas of data in multiple locations, to improve availability and/or performance. They also need to maintain registries of these replicas; these registries are not shown here.

2.6 Transfer Protocols

There are many places in the data architecture where protocols will be used, typically "on the wire", to allow components of the data architecture to communicate, interact and to move data. The data architecture is, in general, agnostic about the choice of transport protocol.

The protocols used for sending operations and notifications between components are those specified in the general OGSA architecture. For *data transfer*, rather than specifying a single protocol, this document allows services to offer a range of protocols. Clients can specify or negotiate which protocols to use for a given transfer operation. (The details of the negotiation mechanism are not discussed in this document).

However, some of the standards that the data architecture is built on may choose to specify specific protocols of interaction, either for performance or interoperability reasons. In general, the data architecture views such specificity as being undesirable but should a group defining a standard decide to be so specific, we do not preclude the use of such a standard in the data architecture.

2.7 Higher-level services

The data architecture allows extra layers of virtualization to be built using the same interfaces as the basic services to hide more complex behaviors as shown in Figure 5. Cache services can collaborate with other services to provide better performance when accessing remote data. Data federation services combine access to data from several data services. Clients may access these composite services in the same way as any other data service without needing to know the details of how they access the underlying data.

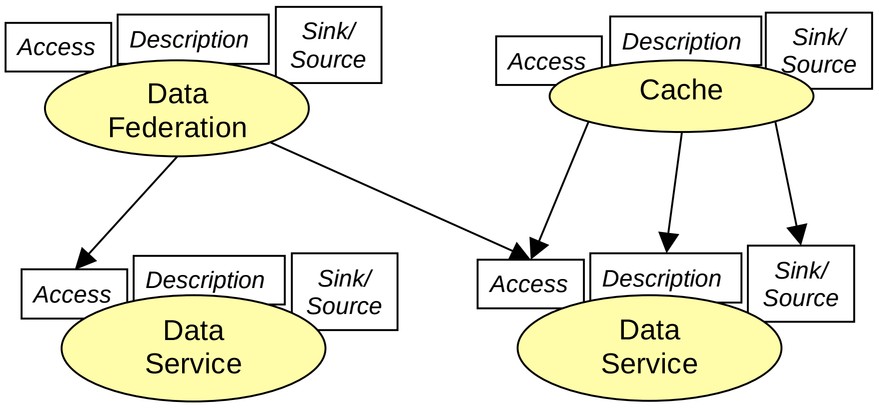

Figure 5: Composite entities in the data architecture

As with the previous diagrams, this is necessarily schematic and abstract. For example, it does not show the data transfer operations between the services involved in a *data federation* or between a *cache* service and its source. Nor does it show how a data federation service will handle a call to its *data sink* or *data source* interfaces. Some services may not offer all these interfaces. As before, generic management and provisioning interfaces have been omitted, as have security services and their related access decision points.

It is worth noting that some resources may provide capabilities such as *caching*, *data replication* or *data federation* natively, outside the OGSA data architecture. For example, several commercial database vendors provide their own replication or federation functionality. To a client of the corresponding OGSA service, it may not be relevant whether the capability is provided in a proprietary manner or by the composition of OGSA services; the implementation details are hidden behind the appropriate OGSA interfaces.

2.8 Virtual Organizations

The OGSA Glossary [OGSA Glossary] defines a virtual organization as follows:

> A virtual organization (VO) comprises a set of individuals and/or institutions having direct access to computers, software, data, and other resources for collaborative problem-solving or other purposes.
>
> VOs are a concept that supplies a context for operation of a Grid that can be used to associate users, their requests, and a set of resources. The sharing of resources in a VO is necessarily highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs.

Virtual organizations are one of the core concepts of grids. Data architecture components are there to help construct VOs. A VO might consist of an entire institution (e.g., a company or a university), a subset of that organization or be composed of many individuals from many disparate institutions. These individuals may be explicitly named or they may be associated with the VO by being part of an organization (e.g., a division of a company, a department in a university) that participates in the VO. The data architecture must be able to support these different types of hierarchies used to construct possible VOs.

Here we briefly mention some areas of the data architecture where VOs must be taken into consideration. The remainder of this document provides details on these and other aspects of the architecture from a VO perspective where appropriate.

- Security. Security, and privacy, of data is critical to the proper operation of a grid and to its acceptance as a computing infrastructure. Classically, security has had its primary focus on security within an institution. The emergence of VOs as a critical component of grids makes it clear that security must be addressed both within an organization as well as within cross-organizational VOs. This has implications both for the architecture of security within a grid as well as how that security architecture interacts with the native security mechanisms of legacy components of the grid and to the security policies enforced within an institution. At the core of these concerns are cross-organizational authentication and authorization.
- Policies. The policies that control operation of a grid must be able to express the needs of a VO. This implies that policies must be institutional, vendor and implementation neutral.
- Storage. The data within a grid has its permanent home within Storage. All participants in a VO have a vested interest in that data. Thus the Storage architecture must meet the needs of those participants. This has implications on security (see above) as well as how data is moved into and out of Storage. Issues of Storage management, accounting/charging must also be properly supported within a VO. These are normally handled within a Storage system to meet the needs of a particular institution. The VO imposes the need to deal with these issues in contexts that may be larger, or smaller, than those classically handled by Storage.

2.9 Policies

Policies are documents that describe or configure the behavior of a service. In the data services, policies may specify the availability, performance, consistency or other aspects of the services. They may specify who can access data, the kinds of access allowed and any restrictions on the transfer of that data.

The quality of service provided by data sources is an important subset of the policies that must be reflected in the data architecture. They include those related to performance, reliability, and data consistency.

Performance is a critical real world requirement for effective use of the Grid by real world applications. Performance properties include, but are not limited to: expected response time to an access or update request, throughput for data transfers, and number of access requests that can be handled per second.

Reliability properties reflect the ability of the data service to operate over long periods of time. For instance, what is the expected time between outages of the data service, the expected time a service will be down, the amount of data that might be lost if an outage occurs and the ability of the data service to continue operation in the face of disaster (e.g., the data center holding the data service is vaporized).

Security is a key consideration in Grid systems. Security policies specify who may access particular data, the locations to which it may be moved and under what constraints. Section 4 discusses security considerations in more detail.

Finally, there are many cases in which changes to one data service may need to be reflected in an associated service. Examples discussed within this document include data replication services, cache services and data federation services. The system will automatically maintain some degree of consistency between the data stored in the base and the data presented by the derived data services with this consistency maintained in the presence of updates to the base. The properties that describe the degree to which base and derived data services are kept in synchrony are called the data consistency properties.

The following is a suggested list of policies to govern data service operation. These policies will need to be formally defined and specified by an as yet to be determined standards body.

- Throughput: the number of access requests that can be satisfied per unit time.
- Response time: the time allowed to satisfy an access request.
- Availability: the percentage of time that the service must be up.
- Recovery time: time allowed for the service to recover from a failure.
- Data resiliency: a specification of the effort the service should make to ensure that data is not lost in the face of failures.
- Access accuracy: if the primary source of some data is not available, how should the service act? Report an error? Find an alternate source (with potentially stale data)? Return partial answers? Another term for this policy might be degradation of answers.
- Currency of data: can the data used by the service, and the data returned, be out of date? If so, by how much?

Services that support one or more of these policies are responsible for enforcing these policies. Note that this list must be extensible so this list is not complete.

2.10 Storage

Data resources, data services and other services described in this architecture will consume storage. This storage might be for data that represents the temporary, dynamic state of the resource. This storage might be long lasting, representing the persistent data of interest to users of the Grid. It may be long lasting data that represents persistent state maintained by and for the operation of the service.

It is up to each service to describe what requirements it has on the storage it will use for its transient and persistent storage. A service may choose to provide means as part of its creation and management interface to allow client control of the storage spaces used by the service. In the remainder of this document we call out specific places where a data service must consider how it will determine what storage to use.

2.11 Denotation of Architectural Components

There are a number of places in this document where it is necessary to name non-addressable entities that are part of the architecture. For example, there needs to be a means to name the query languages supported for access by a data service, or the protocol used by a data transfer service, or the access interface to be provided by a newly created service. These entities will be named by URIs (Universal Reference Identifiers [URI]). We do not mandate how these URIs are generated aside from insisting that each URI should denote a unique entity. We recognize that URIs will be generated in a distributed manner by different standards groups, by different vendors and even by different clients of the data architecture. To avoid duplication of these names, and to foster reuse of previously named entities, we encourage OGF to create a centralized registry of URIs where the URIs for architecturally significant entities can voluntarily be registered.

2.12 Summary

This section has provided an overview of some key aspects of the data architecture. The next section provides an overview of the OGSA architecture which is the context for the data architecture. These two chapters provide the background for reading the remainder of this document.

3 Architectural Context

This data architecture works within the framework provided by the Open Grid Services Architecture (OGSA), which is in turn built on Web Services. This section is intended for readers who are not familiar with the basic ideas of the OGSA architecture. It gives a brief overview of the elements of the OGSA architecture and underlying Web Service specifications that are particularly relevant to the data architecture. For more information about these areas, readers should consult the OGSA architecture document [OGSA].

The OGSA architecture sets out to construct interoperable, portable, and reusable components and systems used to construct VOs in dynamic and heterogeneous environments. Thus, the architecture supports resource virtualization, common management capabilities, and resource discovery, all using standard protocols and schemas. It also aims to achieve resource sharing across organizations, to which end it supports a global naming system, metadata services, site autonomy and the collection of resource usage data. It also makes explicit quality of service requirements and agreements.

- Web Service specifications provide the default messaging layers and service specification languages for a service-oriented architecture. The OGSA architecture builds on these foundations with specifications for, amongst others:

- naming
- management of distributed resources
- security
- notification of events
- resource discovery
- policies and agreements
- reservation and scheduling

The OGSA data architecture describes particular data-oriented interfaces to resources and services within this overall framework. It also specifies dependencies on, and specializations from, other interfaces in the general architecture.

3.1 OGSA Profiles

The OGSA WG follows the lead of the (WS-I)[1] organization by defining normative *interoperability profiles*—guidelines for ensuring consistent and interoperable use of selected specifications. By developing a comprehensive and consistent set of OGSA Profiles that together address all of the required Grid capabilities, the OGSA WG will eventually produce a normative definition of the OGSA architecture.

The first Profile is the OGSA WSRF Basic Profile [OGSA WSRF], which is based on the WSRF and WSN family of specifications. These describe mechanisms for defining and accessing properties, managing lifetimes, and sending notifications. Equivalent profiles based on the WS-Management specifications [WS-Management] may be developed as a parallel activity. OGSA documents treat the underlying mechanism as orthogonal to the properties that are exposed. Thus an alternative architecture could replace the mechanism without affecting the specification of what information is made available. The architecture may be implemented on different underlying infrastructures simply by varying the choice of profile – although of course this will restrict interoperation between such implementations.

The OGSA WG is also specifying the OGSA Basic Security Profiles **Error! Reference source not found.Error! Reference source not found.**. As the OGSA WG agrees other such basic profiles, we expect that future versions of the data architecture will embrace them. Indeed, a future normative specification of the OGSA Data Architecture will take the form of a profile (in contrast to the current document, which is an informational document). The OGSA Profile Definition [OGSA Profile Definition] provides guidelines to be used when developing Profiles.

[1] http://www.ws-i.org.

3.2 Naming

There are many reasons why we need to name entities in a grid. For example, we need to uniquely identify services that we interact with; to log operations for auditing purposes, to map data objects to storage, to map abstract names to (possibly multiple) physical locations, to persist entities in long-term storage, to record the provenance of data, to catalog and search for entities, and many other tasks besides.

Within the OGSA Data Architecture a large number of entities, such as services, resources, databases, results, etc. require naming. These may include, but are not limited to:

caches	namespaces	roles
catalogs	naming schemes	schemas
content identifiers	networks	schema mappings
data formats	people	security contexts
data streams	policies	security tokens
database tables	queries	service level agreements
databases	query result row sets	service types
file directories	references	services
file locations	registries	storage (space)
files	replicas	transactions
identities	repositories	transformations
languages	resolvers	transport protocols
locales	resource locations	user defined entities
metadata	resources	vocabularies

The OGSA work on naming recognizes three levels of name: human-oriented, abstract names, and address. From the OGF OGSA Glossary [OGSA Glossary]:

- **Name** – is an attribute used to identify an entity. In OGSA naming, there are three types of names: human-oriented names, abstract names, and addresses:
    - **Human-oriented name** – is based on a naming scheme that is designed to be easily interpreted by humans (e.g. human-readable and human-parsable).
    - **Abstract name** – is a persistent name suitable for machine processing that does not necessarily contain location information. Abstract names may be dynamically bound to addresses.
    - **Address** – specifies the location of an entity.

And additionally,

- **Resolution** – Name resolution may occur at two levels:
    - Human names may be dynamically mapped to abstract names; and
    - Abstract names may be dynamically mapped to addresses. It is this address, and only this address, that allows messages and operations to be directed at the named entity.

From a data services point of view, the ability to attach names (where possible both human readable and globally unique) to data resources is of key importance. It enhances readability of Grid applications and commands, provides flexibility of use and configuration of applications, and enhances the user experience.

There are a number of requirements on a naming scheme. For example, it should be autonomous, scalable, distributed, secure, reliable, trusted, and have global scope. In addition it is desirable that the naming scheme (and name resolution service) should be fast, efficient, extensible and be capable of being internationalized. It should also be remembered that there will be a requirement to name data that is being generated on the fly, as well as data that has already been materialized and stored. A naming scheme that is not practical to use and does not include these properties is less likely to gain widespread use.

3.2.1 WS-Addressing

The lowest level of naming is the notion of an endpoint name. The OGSA architecture uses a WS-Addressing endpoint reference (EPR) [WS-Addressing-core] to refer to a specific Grid endpoint. Because these endpoints can be highly dynamic in time and space (changing as resources migrate, fail and restart, etc.), thus changing the mapping from abstract name to EPR, it is expected that a naming scheme or any other binding agent will also include some sort of run-time bind and rebind semantic on top of this. WS-Naming [WS-Naming] provides such a mechanism for rebinding EPRs, as well as a mechanism for including abstract names within EPRs.

As noted in Section 1.2, many existing data resources, such as database management systems, have established names and lifetimes that are not necessarily accessible via the OGSA naming scheme. This may lead to an alternative mechanism for naming these resources.

3.2.2 WS-Naming

WS-Naming is a profile on top of the WS-Addressing specification, where additional elements *EndpointIdentifier* and an optional *ReferenceResolver* are included in the WS-Addressing Endpoint Reference. EndpointIdentifiers provide a globally unique and static way of talking about specific resource endpoints, allowing Grid applications to compare and identify resources endpoints for the lifetime of that endpoint and beyond. It should be noted that the set of entities to be named by abstract names is enormous and constantly growing. Thus it is imperative that the mechanism used to generate abstract names scale appropriately.

Note that a WS-Name may specify a resolving service or their syntax may imply a resolving service. This name resolution service provides the mapping between the abstract name and the EPR (or EPRs).

3.2.3 Directory Services: RNS

At the topmost level are “human-oriented names” and as the name suggests these are intended to be read and used by people, and contain structure that is meaningful to humans. They are a primary interface for users and applications. Many of the names will be chosen by people; services may also generate human-oriented names. These names are not guaranteed to be either unique or static.

The Resource Namespace Service (RNS) addresses this human-readable level of naming [RNS]. It encompasses a multi-faceted approach for addressing the need to access resources within a distributed network or Grid by way of a context-specific name that ultimately resolves to a meaningful address, with a particular emphasis on hierarchically managed names that may be used in human interface applications. Its inception is largely based in a file system realm but it is also intended to facilitate namespace services for a wide variety of Grid applications and can be employed to manage the namespace of federated and virtualized data, services, or effectively any resource capable of being referenced in a Grid/web environment.

Mappings between the human names and EPRs are maintained and accessed by the RNS services. These EPRs may be WS-Names, if they include EndpointIdentifiers (i.e. they conform to the WS-Naming specification), or they may simply be addresses.

3.3 Management of distributed resources

OGSA is in the process of defining an information model that describes a wide range of resources that may make up a Grid. These include databases, storage systems, files, catalogs and *data sets*, as well as jobs, processors, networks and others. This information model forms the basis of the OGSA management services.

These management services will be built on suitable Web Service specifications. They will provide operations for querying the status of resources and updating them as necessary. There are two proposals for management of resources in a Grid. The WS-DM standards from OASIS [WS-DM] provide one set of management services. The WS-Management proposal from Microsoft [WS-Management] provides a second. Management facilities depend upon an information model to describe the services being managed. The information model for data services needs to be defined by an as yet to be determined working group, perhaps the OASIS CIM TC, so that the OGSA management services can manage data services.

3.4 Security

Security is a key aspect of Grid systems. Businesses demand it, customers and consumers increasingly expect it and there is a growing set of government regulations worldwide that mandate security. Thus the data architecture must support security and privacy mechanisms that meet client needs in these areas. The OGSA architecture uses and extends security specifications for encrypting data, authenticating users, identity mapping, authorizing operations, delegating access rights, secure logging and maintaining privacy.

Throughout this document we discuss security as it applies to data services. Section 4 will cover general security issues, describing existing work that we build on and also identifying areas where work remains to be done. Security as it applies to specific parts of the architecture is discussed in the relevant sections.

3.5 Notification of Events

In a dynamic Grid environment it is critical that components can request and receive timely notification of changes in one another's states. The OGSA architecture specifies the use of suitable web service specifications that provide this functionality.

These services may be used by data services in many ways. For example, they may be used to notify clients and other services about management events, performance and resource issues, and to implement data consistency mechanisms. They may also be used to externalize database triggers.

3.6 Resource Discovery

Discovery services are vital to the data architecture. The data services may use the discovery services not just for registering services themselves, but also for registering the data that are stored by those services. This requires languages or ontologies for describing data. Discovery services may also register the locations of schema definitions. Discovery services may be built upon metadata repositories – repositories that contain information about other entities in the Grid such as resources and services. Some discovery services may return the name of the service and some may return an EPR to the service. Some discovery services may take a description (e.g., much like a query) of the desired data as input and return the data itself.

3.7 Policies and Agreements

As described in section 2.8, policies are documents that describe or configure the behavior of a service or resource. The OGSA architecture will specify a suitable format for the definition of policies[2]. The details of the policies that a service accepts are part of the specification of that service.

Policies are used in two ways in the OGSA architecture. A policy can be provided by a data service to describe its quality properties. This allows clients to choose services according to their needs. Alternatively, some data services may allow a client to use a management interface to request that the service provide a Quality of Service (QoS) according to a policy of the client's choosing. The service may abide by that policy, reject that policy or engage in

[2] WS-Policy is an example of a policy mechanism but is not, as of this writing, within a standards process.

a negotiation with the client to find a mutually agreeable policy to govern the service's operation. In both cases the same policy description applies.

Agreements are time-limited contracts between a service and a client, or between a group of services and/or clients. They may, for example, state that a certain policy will apply to a given operation. The OGSA architecture will specify a format and negotiation protocol for agreements. A possible candidate is the proposed WS-Agreement **Error! Reference source not found.** standard.

Quality of Service is a key area where policies and agreements are used in the data architecture. QoS will typically be specified via policies that are at the heart of the agreements that will be agreed to between clients and data services to govern the interaction of the client and the data service.

3.8 Provisioning

To automate the complicated process of resource allocation, deployment, and configuration, it must be possible to deploy the required applications and data to resources and configure them automatically, if necessary deploying and re-configuring hosting environments such as the operating system and middleware to prepare the environment needed for job execution. It must be possible to provision any type of resource not just compute resources, for example, network or data resources.

3.9 Execution Management Services

The Execution Management Services (EMS) in OGSA control the scheduling and placement of units of work on appropriate services and thus the resources they represent. Their functionality generalizes the notion of executing a compute job; suitable implementations may schedule any unit of work, such as a database query or a data transfer. EMS services use data services in order to stage the necessary data to the execution server or to access that data remotely. Data services may also be required to provide necessary information to enable these services to produce satisfactory schedules.

An *Execution Planning Service* is a service that builds mappings ("schedules") between jobs and resources. The service will typically attempt to optimize some objective function such as execution time, cost, or reliability.

A *Candidate Set Generator* determines the set of resources on which a unit of work can execute. It typically provides input to an Execution Planning Service.

A *Job Manager* is a higher-level service that encapsulates all of the aspects of executing a job, or a set of jobs, from start to finish. Examples include queue managers, portals or workflow enactment engines. Jobs are specified using the Job Specification Description Language [JSDL].

Currently, more work is needed to integrate the scheduling services with data services. Scheduling services can use monitoring information such as bandwidth, utilization patterns and packet size to choose the best approach for moving a given *data set* to suit the agreed quality of service. Conversely, the EMS may need to request that data services reserve capacity to ensure that a job will be able to execute and achieve its (optimization) goals.

3.10 Reservation Services

A *Reservation Service* presents a common interface to all varieties of reservable resources on the Grid. Reservable resources could include (but are not limited to) computing resources such as CPUs and memory, graphics pipes for visualization, storage space, network bandwidth, special-purpose instruments (e.g., radio telescope), etc.

Currently, OGSA describes the reservation services as one of the EMS services. This needs to be generalized to cover all OGSA services. Data services are likely to need to reserve certain resources in order to operate. For example, a data transfer operation will require storage space and network bandwidth, while a *data federation* service may require compute power in order to perform join operations. Conversely, the EMS may need to request that data services reserve capacity to ensure that a job will be able to execute and achieve its (optimization) goals. Similar considerations apply when the provision of a data service must be scheduled for a certain time.

3.11 Transactions

This architecture does not define a mechanism for distributed transactions per se. We expect this functionality to be provided by other developments in the Web Services community. Currently there are two families of specifications under development. On the one hand, there is WS-Coordination [WS-Coordination] and two of its coordination types, WS-AtomicTransaction [WS-AtomicTransaction] and WS-BusinessActivity [WS-BusinessActivity]. On the other hand is the WS Composite Application Framework family (WS-Context [WS-Context], WS-Coordination Framework [WS-Coordination Framework] & WS-Transaction Management [WS-TransactionManagement]).

Although we do not define a transaction mechanism, nor do we choose from existing ones, we do need to ensure two things. First, that the transaction mechanism satisfies the needs of data. Second, we must ensure that whatever transaction mechanism(s) we endorse properly flows through the architecture.

Both transaction systems noted in the first paragraph of this section meet the known needs of data. They have been developed by the relevant communities and have been endorsed by various database vendors which gives us assurance that they have adequate functionality for the purposes of data.

In both cases, the transaction specification depends upon carrying a transaction context along with every port call. It is the responsibility of the implementation of each port to honor the transaction context, or to ignore it. There is no need to explicitly change the signature of a port to accommodate transactions. Thus, for the purposes of the data architecture, we assert that transactions layer on top of the basic data architecture in a transparent fashion. We strongly suggest that the descriptive information for the implementation of a port include information on how it handles this transaction context.

Finally, it must be observed that it is up to the clients of a particular port to decide if the decision of a particular implementation of that port to honor transactions, or not, is sufficient for the needs of that client.

3.12 Sessions

Operations on data tend to be asynchronous. Applications also tend to issue sequences of operations against data sources. Both of these argue for the creation of the notion of a *session* to contain the context for the interactions between a client and a (set of) data services.

A session should allow a client to start an operation at a data service and have control immediately return to the client. At a later time, the client can use the session to ask the data service about the status of its previously issued request(s).

Sessions can also be used to optimize certain sequences of interaction between a client a (set of) data services. For example, security requires that every requestor of an operation on a data service be authenticated and that the particular operation being requested be authorized for that requestor. Doing this on every operation request could be quite expensive. A session provides one means for doing this authentication and authorization once and, in essence, caching the result for the duration of the interaction between the client and the service. Other examples might include quotas on uses of resources at the data service and reservation of resources at the data service.

These examples show that the notion of session will be vital to creating a well performing data architecture. At the moment we are not aware of any work going on in OGF or other standards bodies to define a session mechanism.

4 Security

Security is pervasive and must be addressed at all levels of a Grid system, both within an organization and in each Virtual Organization (VO) that the organization participates in. Every organization that is a member of a VO will have its own security policies. It is important that the Grid that is used as the fabric to compose a VO does not weaken any of the existing security policies of any of its members. Failure to properly support security at any level of a Grid system raises the prospect of compromising the security in other parts of that Grid system. Therefore to ensure security of the data architecture, security must be ensured in all parts of a Grid system.

There are environments, such as within an organization, that are inherently secure. Inside these environments it may not be necessary to impose security on: communications, authentication of users, authentication of requestors of services or the enforcement of access checks. However, in many organizations, and in almost all VOs, it is still necessary to impose these security controls.

A VO is an important case from a security perspective. Within a VO, the security of the VO is only as strong as its weakest member. An organization participating in a VO should make security checks at the boundary of the organizational environments. This is necessary since one must treat the external environment as being inherently insecure, see Figure 6.

The remainder of this section describes the mechanisms needed to achieve privacy and security goals both within an organization and within a VO. It should be noted that there may be environments that are partially secure. For instance a VO may be such an environment if security enforcements are carried out at the VO's boundary. In such environments, some or all of the security mechanisms may not be needed to achieve specified security and privacy goals.

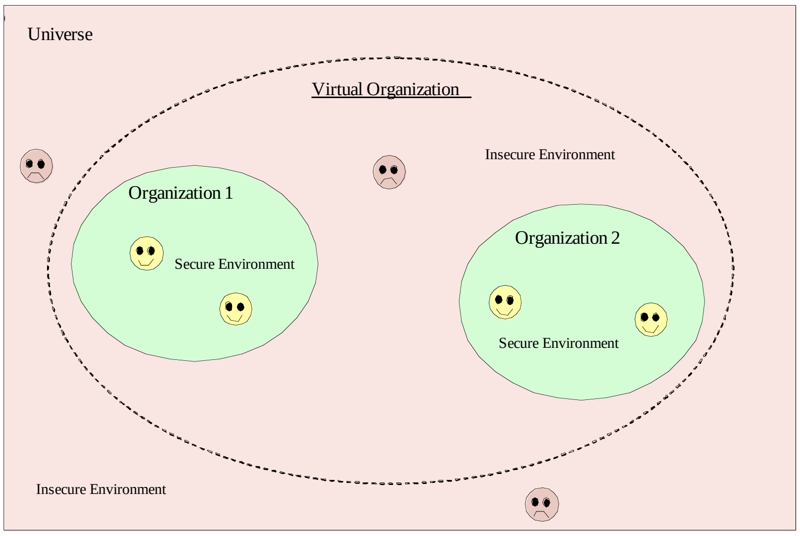

Figure 6: Representation of the different domains and the possible security implications

Security begins at the communication level of the system. This is important to secure communications within an organization, between organizations within a VO and with organizations outside the VO. The "OGSA Basic Security Profile – Secure Channel 1.0" **Error! Reference source not found.** describes a security profile that ensures secure, authenticated communication between clients of web services and the web services. The OGSA WG is also preparing an alternate security profile that is applicable in environments where secure communication is not necessary. We would expect any OGSA profile for data services to require compliance with one of these profiles.

Authentication of users is normally a prerequisite for proper authorization to use a data facility. The secure channel profile (see above) ensures that *services* are properly authenticated. Data requires that clients of the data facility be properly authenticated. Thus use of a standard authentication mechanism such as that specified by the basic OGSA Basic Security Profile is essential.

Each organization or resource may haves its own authentication system. In communicating with the outside world, it may be necessary to map between the shared identity of the VO and the corresponding identity within the resource or organization. For example, an X.509 digital certificate may be mapped to a local username. Translation of the identity of an invoker of an operation to an identity known to the internals of a data service, the data resource, will be a key requirement in many data service implementations. The identity federation and translation mechanisms provided as part of the overall Grid security mechanisms should be designed so that they can be used to address this need.

Once a user is authenticated, the user must be authorized to perform the task being requested. This authorization has two aspects:

- Is the user allowed to invoke the operation to perform this task at all?
- Is the user allowed to perform the indicated task on the data denoted, directly or indirectly, by the parameters to the operation?

The former is no different for data than for other aspects of the overall Grid architecture. Any security mechanisms defined for operation access apply to the data architecture with no additional requirements.

Authorization to perform a task on a specific set of data is a requirement that is above and beyond the authorization requirements for operation invocation. For instance a user may be allowed to query some tables in a relational database but not all. A user may be able to open some files for reading, some for read/write and some not at all. Due to the generality of operations in the data architecture, the implementation of a particular operation must be able to use the authentication information supplied as part of the operation invocation and basic Grid system authorization mechanisms as *input* to a decision making process to determine if a particular operation on a particular set of data is permissible. For example, it may be the case that the VO authorizes a user to invoke a particular operation but the resource then blocks access to particular data. The resource always has the final say in whether a particular action is permitted. .

Authorization must also take into account user roles. A particular user may be able to perform administration operations when acting as a system administrator but would not be able to access certain data. That exact same user, when acting as, say, a payroll specialist would be able to access payroll related data but would not be allowed to perform system administration functions. Thus the basic Grid system authentication and authorization mechanisms should support roles.

In some environments (e.g., government) there is a requirement for non-discretionary, multi-level, access controls (e.g., categorizing data as public, secret, and top-secret). Many data systems support this class of access control today. The authorization and authentication mechanisms of a Grid system must, at a minimum, be compatible with this class of access control. Direct support of non-discretionary access control may be required in some environments and would impose yet more requirements on the authentication and authorization mechanisms of a Grid system.

Authorization must also take into account sequences of access requests. It is possible that individual access requests are allowed but the result of a sequence of operations results in unauthorized access to information. The Data implementation must be able to prevent this release of unauthorized information not only from a security perspective but also from a privacy perspective.

Privacy is related to security. There is an increasing body of government laws and regulations around the world that mandate privacy. The data architecture in particular and the Grid system in general, must adhere to these privacy requirements. Some issues/examples that need to be addressed:

1. The set of access requests from a user may need to be private to that user. This impacts the logging of those queries by the data service.

2. Privacy of data needs to be assured when at rest (e.g., on disk or tape). This may require encryption of data when it is at rest.
3. Privacy of data in transit (e.g., the result of a data access request) must be ensured. This may require encryption in the communication channel.
4. A data service should advertise the degree of privacy that it supports. We are not aware of any work being done presently for a vocabulary to express this.

If a data service needs to access a data resource, be it another data service or a non-service data resource, that access will be governed by security and privacy mechanisms. The data service will need to supply the required information so that the access can be authorized. The data service is responsible for determining what security credentials to use for the access. These may the credentials of the application calling the data service or the credentials may be an inherent part of the state of the data service.

When a data service returns data, it must take measures to ensure that the security and privacy characteristics of that data are noted. It will use an as yet to be determined mechanism to attach this information to the returned data. It is beyond the scope of the data service to ensure that this security information is honored. It is the responsibility of the Grid security infrastructure to provide suitable mechanisms for ensuring that these security and privacy controls are honored.

In some environments, the physical location of the entity receiving data is crucial (e.g., areas that are allowed to hold top secret data). This may restrict the ability of the data service to return data to an otherwise authorized requestor. We are not aware of any standards work that addresses this requirement.

Finally, the ability to log and audit actions is an inherent part of a secure system. Thus, not only must a data service enforce security and privacy concerns, it must also take measures to ensure that its actions are auditable (and this generally means that they are logged). A data service must also ensure that the requests of its clients are auditable. Again, this generally means that the requests are logged. The OGSA architecture will specify secure, tamper-proof logging services. These may in turn use data services to store and query the logs. Existing standards work in the logging and auditing areas may need to be extended.

These aspects of security impose requirements in every component of the data architecture. Ideally, every service in an actual realization of the data architecture would actively support and implement the standards that exist, or will exist, for these requirements. However, we recognize that some services may not provide this support. Recognizing this, we strongly suggest that all services:

- Advertise the degree to which they adhere to security requirements.
- Accept security related information in their interfaces. We anticipate that much of such information will come as part of a standard header that is part of every service invocation.
- Pass security related information such as security credentials in all service requests from this service. This security information may be held within the service or may have been provided as part of an invocation of this service.

The above considerations identify a number of areas where we are not aware of work being done. To provide a secure data architecture, working groups need to define the following areas:

- The syntax and semantics of security policies need to be defined. These policies need to be attached to data and also need to be the subject of negotiation between data services and their clients.
- A mechanism is needed to attach security policies to data in motion.
- A means to specify the geographical location of requester and resource is needed. Geographical location must be an optional part of security policies.
- A specification of the reason for access to data needs to be defined. This reason needs to be part of security policies.

- A means to specify authorization of sequences of access requests is needed. As noted above, there are situations where individual access requests are valid but a sequence of requests results in unauthorized release of information.

5 Policies

Policy Operations	Description
ReplacePolicy	Replaces the entire policy governing the operation of the service with the policy supplied as an argument.
UpdatePolicy	Updates part of the policy governing operation of the service

Table 1: Policy Operations

The details of Policy are specific to each (class of) service within the data architecture. There are, however, operations that are common to all services within the data architecture and which should be supported by all of these services. Within this document we assume that all of the services described here provide these operations.

During the lifetime of a service, there may be need to update the policies controlling the execution of the service. We require two operations to change these policies – one to change all of the policies and one to change a subset of the policies. In this formulation, the service may either agree to the changes, or choose to not accept them. In the longer term we would expect that some services would support negotiation of these changes via a standard OGSA agreement negotiation mechanism.

ReplacePolicy() – replaces the entire policy governing the operation of the service with the policy supplied as an argument. Upon successful return from this operation, the old policy governing service operation will have been discarded and service operation is solely governed by the policy supplied in ReplacePolicy. It may require a sophisticated policy engine to allow the service to determine if the requested policy can be met. If the new policy can not be used by the service, an error will be raised and the policy governing service operation will be unchanged. The service must specify the effect of changing policy on currently executing operations.

UpdatePolicy() – operation updates part of the policy governing operation of the service. It takes a set of policy terms to be updated as an argument. Upon successful completion of this operation, any policy terms provided as part of the input replace the same terms previously governing service operation. An error will be raised if any of the new policy terms cannot be honored and none of the policies governing service operation will be changed.

6 Data Description

The OGSA Data Architecture supports heterogeneity in data types and in the resources and services that contain them. Effective discovery, interpretation and association of data in the OGSA environment rely on the availability of suitable descriptive information. This aspect of the architecture concerns the specifications available to describe data itself and the resources that store data.

As previously explained (section 2.3), the data itself is managed by a data resource that is outside the scope of this architecture to define. Access to the data, to the capabilities of the data resource and to the current status of the data resource and the data itself, is provided by a data service and the operations and properties that it supports. It is the data service that provides the architected interface to the data.

The capabilities of these entities are described by properties in the interfaces supported by the service. There are two main classes of properties that describe the data provided by a data resource:

- Properties that describe the data itself, such as its format, encoding, schemas, and provenance.
- Properties pertaining to the data resource, e.g. status information, query languages that it may support, etc., see Figure 7

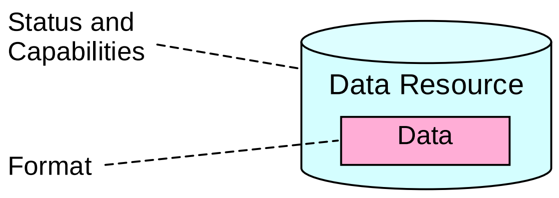

Figure 7: Data description elements

Some of these properties will be read only; others may be settable. Settable properties may affect the behavior of the service interface and thus the way that a client interacts with that service. It is worth noting that the property values presented to a particular client may be affected by the security credentials presented by that client.

The following sections discuss the description of *data formats* and the *data resources* themselves. The data services that expose them to the Grid are described using the standard OGSA mechanisms.

6.1 Format Description

In its electronic form data is ultimately stored as a series of 1s and 0s. In order to make use of the data its encoding, structure, classification and organization must be understood. Here we use the term *data format* in place of these various terms. Only by understanding the relevant format can the data be usefully employed.

Data format descriptions must provide support for:

- Different encodings of data, e.g. binary, text, etc.
- Standard formats, (e.g. JPEG [JPEG], TIFF [TIFF]) and application-specific formats (e.g., a list of comma-separated values in a particular order).
- Data which is stored using specific data models, e.g. relational, XML documents, files, etc. within a storage infrastructure such as a DBMS, XML database, a file system, etc.

There are almost as many data formats as there are *data sets* and often one format relies on another. For example, an XML document that is valid with respect to an XML Schema must also necessarily be well formed with respect to the XML specification [XML] and will be presented in a specific character encoding, for example, UTF-8 [UTF-8]. We need URIs to name these data formats, following the guidelines discussed in Section 2.11.

The format of data may be well known or discovered at run time. Some data is formatted in accordance with well known and accepted standards, For example, a Portable Network Graphics (PNG) file is a binary file structured in accordance with the PNG specification [PNG]. All that is required to describe the format of the file is a simple code. This could take the form of a file name extension, a code in the header of the binary file itself or, for example, a mime type (image/png). Knowing that the file is a PNG file allows the appropriate software to be used in order to interpret the binary file in accordance with the correct PNG standard.

Alternatively, while a well formed XML file abides by the rules of the XML specification, it is only valid with respect to some schema. We might be able to tell an XML formatted file from other file formats by, for example, its file name extension or by its mime type (text/xml). This is not sufficient to understand the valid structure of the contents of the file. In the case of XML this means referring to the appropriate schema document. The name and location of the schema document may be carried explicitly by the XML file. Alternatively this information may be provided implicitly, for example, a WSDL file describes the schema of the XML messages that pass to and from service endpoints. Given a message name the appropriate schema can be implied by looking up the message name in the WSDL file.

In summary two pieces of information are required to describe the format of data. Firstly the well known or static format of the data can be stated. Given this basis any optional dynamic format information can be provided.

In some situations data services will provide this information as part of the resource description, for example, a data service exposing an XML collection implicitly manages XML formatted documents and may list the schemas to which contained XML documents conform.

Alternatively a data service that exposes a file system may simply provide byte IO access and rely on the service consumer to establish each file's format based on factors such as the file name extensions or the contents of the file itself.

6.2 Resource Description

The term *data resource* refers to the entity that holds the actual data. Data resources can take many forms from file systems and computer memory to databases in database management systems and messaging systems. In this architecture data resources may be exposed for management or access reasons by one or more data services.

The description of a data resource is embodied in a series of properties, or annotations. These properties describe the capabilities and status that the data service makes available for each resource, for example:

- Resource name and type
- Ownership and version information
- Capabilities, e.g. max capacity, query languages supported, behavior
- Structure of data within the resource, e.g. a database schema, XML Schema [XSD], DFDL[3]
- State, e.g. lifetime, used and free capacity

There is work required here to define a set of generic data service properties. Currently we are not aware of any standards body looking into this.

6.3 Third-party descriptions

In addition to properties of a data service, some descriptive information may also be made available by third parties such as a registry, see Figure 8. This is often the case when the properties can change dynamically or are generated by consumers who do not own and

[3] DFDL is being developed by the DFDL working group of the OGF. See http://forge.gridforum.org/projects/dfdl-wg.

manage the original data service. The various descriptions may overlap or be distinct. They may be kept in synchrony or be managed separately, depending on the design of the particular system.

Examples of such annotations include;

- Descriptive information, e.g. human readable text
- Classification, e.g. Dublin Core [Dublin Core]
- Relationships, e.g. RDF

This separation of concerns, between the registry and the data resource itself, is useful when, for example, the data is classified by different people in different ways. In such cases, regardless of how many classification annotations are made, the size of the data will remain constant.

Generally this architecture expects that these types of annotation are managed separately from the fixed properties of the data resource. This is not of course a hard and fast rule.

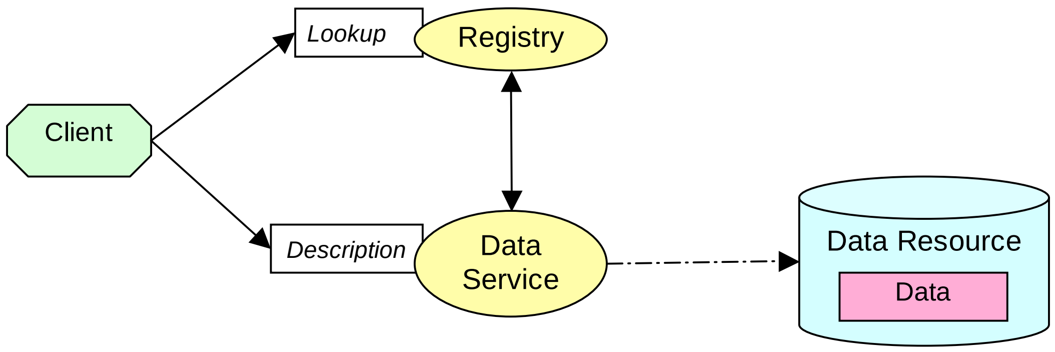

Figure 8: Third-party description of data

When data is created, updated, transferred between resources, or deleted, any corresponding data descriptions may also need to be updated, depending on the policy in force at that time. Scenarios that compose data services to implement a particular functionality must take account of such constraints.

More details about the interface provided by a registry are discussed in section 13.

6.4 Provenance

Users of data services may also wish to see information about the provenance and quality of the data provided by those services. Provenance properties reflect the history of the data. Where did it come from? What processing was performed on it? What software was used to perform the processing? Which human being requested the processing?

Provenance information may be stored at the level of the whole resource or of its component parts, sometimes to the level of individual elements. This in turn requires the services or other processes that generate the data to also maintain the consistency of the provenance information. It also requires an architecture for maintaining this information. Complete provenance information can allow the data to be reconstructed by following the workflow that originally created it. Provenance information may be provided by the service itself, or it may be maintained in a data catalog or a logging service.

7 Data Transfer

The table below summarizes the operations proposed in this section.

Factory Operation	Description
CreateTransfer	Initiates a transfer between a data source and a data sink. Returns an EPR to a Transfer service that is managing the transfer.
Data Resource Operations	**Description**
GetHandle	Returns a WS-Name for the transfer related operations provided by that service
GetSupportedProtocols	Returns the set of transfer protocols supported by this data resource.
SetupTransfer	Returns a protocol-specific URI which can be used to initiate and mange the transfer.
Transfer Service Operations	**Description**
PauseTransfer	Temporarily stops an in-progress transfer.
ResumeTransfer	Resumes a paused transfer.
StopTransfer	Stops an in–progress transfer. The Transfer service is destroyed as part of this operation.

Table 2: Data Transfer operations

The transfer of data is a key aspect of the data architecture. Data transfer is the movement of data from a source of data to a consumer of that data (a data sink) – ultimately, the movement of bytes of data from one computer to another over a network. Data transfer can happen when a data access interface returns a result to a third party, when data is staged to an execution server, when a data replication service copies data to a replica and in general whenever multiple data services need to exchange data.

In the request-response pattern, as described in section **Error! Reference source not found.**, the data to be returned will be described in the message from the client to the data service and returned in a the response message from the data service to the client, using a protocol such as SOAP With Attachments [SwA], DIME [DIME] or MTOM [MTOM]. This form of data access is described by the basic message exchange and operation invocation standards. Typically such exchanges will be used for small amounts of data.

In other cases, the amount of data to be transferred can be quite large. For example, a file containing the results, perhaps terabytes in size, of an experiment at a particle accelerator might need to be staged to a computational node where the raw data is to be analyzed. A database might be hundreds of gigabytes, or even terabytes, in size and need to be replicated to a second site to ensure availability. Whatever the reason, the amount of data may be very large. This class of data transfers is the topic of this section.

OGSA uses Web Service mechanisms to communicate with services – effectively this means using SOAP. However, SOAP is not a good transport protocol for large amounts of data. Thus, a data transfer will employ an alternative means of delivering data other than through SOAP. So, the data access query-response pattern described in Section 2.5 could use an alternative data delivery mechanism, thus coupling with data transfer, to implement the third party delivery (where the delivery could be to the original client itself).

The data transfer mechanism should meet the following requirements:

1. Performance. Data transfers need to operate efficiently.
2. Protocol agnostic. The data transfer mechanism should be able to support various transfer protocols where appropriate.
3. Data transfers may be controlled by a data transfer service when this is appropriate.

4. Data services, other than those managing the transfer itself, should not be required to understand the transfer protocols being used.
5. Data transfer should not be tied to the type of the data. Data might be in many forms (e.g., flat files, relational databases, image files) and the data transfer mechanism should handle the appropriate types.
6. Data transfers should be manageable by the client requesting the transfer. The client should be able to monitor status, adjust performance and terminate the transfer early if needed.
7. Policies and agreements will govern the negotiated behavior of a data transfer initiated by a client.
8. Data in transit will be secured by the data transfer mechanism in accordance with negotiated security goals.
9. A data transfer service may provide the ability to transform data as it moves from source to sink.

The most basic level of data transfer operation simply transfers bytes across the network without interpreting them. Higher-level transfer services may perform transformations on the data in order to preserve the data semantics as it is transferred between services, e.g. swap bytes in going from a little-endian to a big-endian machine.

As part of the data architecture, we require that all data services provide a set of interfaces that allow for the standardized transfer of data between data services (the data source and the data sink). In order to understand these interfaces, we first describe the interactions that must occur between a data transfer service, a source of data and a sink (destination) for data. The Data Movement Interface Working Group (DMI WG) in the OGF is working on a specification for managing data transfer (just moving the bytes) [OGSA DMI]. The authors are collaborating with the DMI WG to ensure that their specifications will fit into the OGSA Data Architecture.

7.1 Data Transfer Interfaces

The data transfer mechanism will have the following aspects:

- Data Source: This denotes a data service that can act as the source of a data transfer operation.
- Data Sink: This denotes a data service that can act as the recipient of a data transfer operation.
- Transfer control: This instantiates and controls a data transfer operation.
- Transfer negotiation: This queries the sources and sinks of a potential data transfer and brokers an agreement about which protocol to use.

Each potential source or sink for a data transfer must provide the following operations:

GetHandle() – returns a WS-Name for the transfer related operations provided by that service.

GetSupportedProtocols () – returns the set of transfer protocols supported by this data resource. It takes the following argument:

- SinkOrSource: an enumeration indicating whether the data resource will be used as a source or sink for the data transfer operation

SetupTransfer () – which takes the following arguments and returns a protocol-specific URI which can be used to initiate and mange the transfer:

- Protocol: an indication of what protocol is to be used for the transfer operation
- SinkOrSource: an enumerated indicating whether the data resource will be used as a source or sink for the data transfer operation

In addition a data service may provide protocol-specific operations to allow the CreateTransfer operation to determine protocol-specific capabilities of the data resource.

There is a CreateTransfer factory operation that initiates a transfer between a data source and data sink and creates a transfer service instance to mange that transfer:

> *CreateTransfer ()* – takes the following arguments and returns an EPR to a Transfer service that is managing the transfer:
>
> - Source: a WS-Name for the source of the transfer
> - Sink: a WS-name for the destination (sink) of the transfer
> - QOS: An indication of the quality of service required of the transfer operation. This might indicate, for instance, the required time of delivery, desired transfer speed, maximum bandwidth to be consumed and the maximum cost to be incurred.
>
> This operation will use the GetSupportedProtocols operation to determine an appropriate transfer protocol between the indicated sources. It will then use SetupTransfer to create the actual transfer and begin the actual movement of data.

The sink for a transfer operation must be large enough to hold the data to be transferred. A CreateTransfer realization may check that the sink provides enough storage for the data to be transferred. For example, if a file is to be transferred, the amount of data to be moved is known up front. If there is insufficient storage, CreateTransfer may either raise a fault or ask the sink to increase the size of the destination (if this fails, a fault will be raised). The interfaces to do these activities are not specified as part of this architecture as it is inherently dependent upon the exact nature of the source and sink. We observe that for some sources it may not be even possible to determine the size of the data to be transferred before the transfer begins (e.g., data generated dynamically by computation), implying that CreateTransfer may not raise an insufficient space fault but the transfer itself may fail due to insufficient space..

A Transfer service provides the following operations to allow clients to manage a transfer operation.

> *PauseTransfer ()* – temporarily stops an in-progress transfer. A fault is raised if the transfer is not currently in progress.
>
> *ResumeTransfer ()* – resumes a paused transfer. A fault is raised if the transfer is not in a paused state.
>
> *StopTransfer ()* – stops an in–progress transfer. The Transfer service instance is destroyed as part of this operation.

In addition, we suggest that a Transfer service provide operations to determine the current state of a transfer, how much progress has been made and how much data/time remains for the transfer.

A Transfer service should also provide properties describing the state of the transfer. The following properties are suggested:

- Amount of data transferred so far in units specified by the transfer instance.
- Total data to be transferred (if available) in units specified by the transfer. instance
- Transfer protocol being used.
- Rate of transfer so far.
- Expected time to completion of transfer.
- Average network bandwidth consumed so far.
- Maximum network bandwidth consumed so far.

In addition, we need URIs to name data transfer protocols, following the guidelines discussed in Section 2.11. Some such work has been done by the OGF's ByteIO working group, producing URIs such as:

- http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple
- http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/dime

- http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/mtom

Further URIs are required for other protocols, such as FTP [FTP] and GridFTP [GridFTP] among others.

8 Data Access

Data access defines a generic set of service interfaces through which a client can extract data from, or send data to, some underlying resource.

The tables below list different types of data access operations. These are divided into three broad classes:

- A generic set of data access operations that can be used regardless of the underlying data model. These use the factory pattern to create or associate a relationship between a resource and a service with a specific interface.
- A set of data access operations that act on "unstructured" data, i.e. where one is only interested in reading or writing uninterpreted bytes.
- A set of access operation that act on structured data, such as that stored in relational databases, which can be queried or modified using an appropriate query language, e.g. SQL, XQuery, etc.

Factory Operation	Description
Create	Create an association between a data service and an underlying resource with a given service interface. The underlying resource may be created and populated as a result of this operation.

Table 3: Generic data access operations

Access Operations	Description
Read	Takes an offset and a number of bytes to read, and returns that number of bytes from the resource.
Write	Takes a fixed number of bytes of data which it writes to the resource.

Table 4: Data access operations for unstructured data

Access Operations	Description
ExecuteQuery	Provide a means for querying a resource. This could take an SQL, an XPath or XQuery expression or any language type described by the LanguageMap property which is then run by the underlying resource.
BulkLoad	Provide a means of inserting large amounts of data into an underlying resource.

Table 5: Data access operations for structured data

These interfaces and properties are described in more detail below.

8.1 The Generic Data Access Interface

These operations facilitate the provision of data access and implement some of the access patterns described in Section **Error! Reference source not found.**.

Create() – provides a means of associating a service with a resource with a specified interface. The resource may be created and populated with data from another resource, e.g. the result of a query to a relational database, data from files or data pushed to the service by a client, etc.

- DataResourceAbstractName – the abstract name of the resource the message is targeted at.
- PortTypeQName (optional) – the QName of the service interface through which the resulting data should be accessed. This must be one of the QNames found in the ConfigurationMap property which MUST be there when the Create operation is present in the interface.
- ConfigurationDocument (optional) – a document that specifies the initial value for the properties of the resource that is to be used to access the data.
- PreferredTargetService (optional) – the EPR of the preferred service that is to act as the host for the new resource.
- Recipe (optional) – a means of describing the data that is to be used to populate the resource. This could be a SQL query, an XPath expression, a file name, etc. If the Recipe is not present then a service will be associated with the interface specified by the QName

The response message should contain an EPR to the service through which the given resource will be available or a fault will be included.

For each data resource accessible through a service the following properties will be available through the service:

Readable – Boolean indicating whether the underlying resource is readable.

Writeable – Boolean indicating whether the underlying resource can be written to.

ConfigurationMap – an optional property which provides a means of enumerating the supported interfaces, represented as QNames, that can be used by a services to access a derived resource. The desired QName can then be specified in the arguments specified in the Create operation. This property MUST be present when the Create operation is in the interface otherwise it is optional

8.2 The Unstructured Data Access Interface

This interface has the following operations:

Read() – provides a means of reading a specified number of bytes of data from a given offset and this is returned to the client. The operation MAY respond with fewer bytes than requested by the operation. Arguments:

- Start-offset - the offset (an unsigned long describing the number of bytes) into the data resource at which the client wishes to begin reading.
- bytes-per-block- the number of bytes in a single block that the client wishes to read.
- num-blocks - the number of blocks that the client is reading.
- stride - the offset or delta describing how far apart the beginnings of each block of data are inside the data source.
- encoding - a URI specifying the encoding to be used to return the data.

Write() – provides a means of writing a specified number of bytes after a given offset and returns the status of the operation back to the client. Arguments:

- start-offset - the offset into the resource at which to begin writing the block of data.
- bytes-per-block - the number of bytes of data that are to be written for each block of data.
- Stride - the number of bytes that separate the beginnings of each block in the data sink. Blocks are considered to be written sequentially in the order indicated by the stride (i.e., if the stride is positive, then the sequence is in ascending absolute offset order whereas if the stride is negative, the presumed order is descending absolute offset of the blocks). The implication here is that the offsets of the blocks are calculated in isolation from the actual block sizes. If the stride is larger than the block size, then the blocks will be written to the file leaving holes in the middle (the behavior being that these

holes should retain any values present prior to the write request if applicable, otherwise the values in the holes are undefined). If the stride is less than the block size, then blocks will overlap and the final value of any given written byte will be that of the last block in the sequence to overlap that byte. Regardless of the stride value, blocks are assumed to be concatenated directly together in the write requests data block (i.e., stride does not apply to this block).

- encoding - a URI specifying the encoding to be used to send the data

The interface has the following properties:

Size - The total size (in bytes) of the resource.

DataEncodings - A list of URI's indicating the encodings supported by the service for sending or returning the data. If an encoding is listed here, then the service MUST support that encoding as per the appropriate specification.

CreateTime - The timestamp for when this resource was created (relative to the hosting environment). Creation time is loosely defined as the time at which a client could first use any of the methods (or access any of the properties) on a resource via the service and receive a valid, non-error response message. It is left up to the implementer to determine what this time should be.

ModifiedTime - The timestamp for when this resource was last modified (relative to the hosting environment). Modification time is defined as any time after which any client having previously read a block of data (with the read method), or having accessed the Size property, could now expect to receive different results were it to call the read method or access the properties again.

AccessTime - The timestamp for when this resource was last accessed (relative to the hosting environment). The last access time is defined as the last time when any of the access methods in this specification were last called.

8.3 Structured Data Access Operations

This interface has the following operations:

ExecuteQuery() – provides a general form of performing query like operations which allows an expressions consisting of any of the query languages specified in the LanguageMap property which passed through the interface and run by the underlying resource. Data may also be passed to the underlying resource in the message to insert or update already in the resource. Arguments:

- DataResourceAbstractName – abstract name of the resource at which the message is directed.
- DatasetFormatURI – An OPTIONAL element that MAY be used to define the format of the response dataset. This element, when present, MUST contain a URI from the set that appears as DatasetMap property elements. When this element is omitted the format of the response message will follow the format referenced by the first DatasetMap property.
- GenericExpression – the query expression document. The document MAY contain URI attribute indicating the language used to form the expression. The URI of this attribute, if present, MUST be one of those specified in the LanguageMap property described below. When no language attribute is provided the first LanguageMap entry for GenericQuery is assumed.

BulkLoad() – provides a means of inserting large amounts of data into an underlying resource.

This interface has the associated properties:

- *LanguageMap* - list the valid query languages that can be used in the ExecuteQuery interface. A query language is represented by a URI which is mapped to a QName representing the language.

- *DatasetMap* – enumerates the supported data formats in which data can be returned to the client. The valid return data formats are represented by URIs which are mapped to QNames. A client can thus specify the format in which they want the data returned in the access operation by specifying the corresponding QName.

8.4 Conclusions

These data access interfaces need to be flexible and be able to cope with data which varies in its size and granularity, i.e. one item that is a gigabyte or a thousand items of a megabyte. Moreover, when using web services and non-trivial amounts of data alternative delivery mechanisms may need to be considered and data transfer facilities coupled with the above primitives to transfer data to a resource or to return data to a client or consumer. The usage patterns described in Section **Error! Reference source not found.** may be used to create composite entities that achieve this.

In addition, access mechanisms may provide different levels of abstraction of the underlying data model and formats used. The level of abstraction offered by a service to the underlying data will vary according to the capabilities and performance required by a consumer, so if required the user may not know whether the data that they are retrieving is stored in a relational database or a file system – clearly though these considerations would have an impact on performance. As a general principle though, the data access interfaces proposed here are not intended to supplant existing native forms of access but rather to operate in conjunction with and leverage them. It is important, if these access mechanisms are to be useful, that they should add little overhead over native access mechanisms provided by the underlying resource.

The data architecture currently describes two specifications for data access: WS-DAI and ByteIO produced by the DAIS and OGSA ByteIO Working Groups of the OGF. How the generic operations described in this section map to these specifications is considered in Section **Error! Reference source not found.** and **Error! Reference source not found.** respectively. Other types of data access interfaces may be added as the need arises.

9 Storage Management

Space Management Operations	Description
ReserveSpace	Facilitates negotiation of space reservation.
GetSpace	Returns space tokens for currently allocated spaces.
ReleaseSpace	Releases an occupied space. Depending on local settings and policies, all files may need to be released in the specified space before the space can be released.
Directory Management Operations	**Description**
ListFiles	Returns a list of files and information about them.
ReleaseFiles	Releases the files in a storage space. The files are not necessarily deleted immediately, but the space occupied by the released files is eligible for removal if space is needed.
RemoveFiles	Removes files from a storage space.
CopyFiles	Copies files, either within a storage space or between (local) spaces.
MoveFiles	Moves files either within a storage space or between (local) spaces.
MakeDirectory	Creates a directory in a local storage space.
DeleteDirectory	Deletes a directory in a local storage space.
Transfer Management Operations	**Description**
OrderTransferFromStorage	Reorders a transfer request from a storage resource based on the current known state of that resource.
OrderTransferToStorage	Reorders a transfer request to a storage resource based on the current known state of that resource.

Table 6 Storage management operations

Along with computing and networking resources, data storage resources are one of the basic building blocks of a distributed computing infrastructure. There are many kinds of storage resources to consider, ranging from a memory stick to a multi-petabyte tape silo. Different storage resources offer different levels of Quality of Service, and have different semantics for data access, both for reading and writing.

In a Grid environment the data and its availability is managed by the Grid middleware. The middleware interfaces to storage need to be rich enough to be able to take into account the possible semantic properties of the underlying storage hardware. At the same time its usage should be simple enough so that clients of the storage do not need to be experts in the handling of storage.

The storage resource provides space to store data. That space may be used for storing files or it may be used as a raw device that can be formatted and mounted or used as a block device. The basic 'resource' that storage has to offer is therefore storage space. The space is managed through the management interface provided by the storage service.

9.1 The Storage Resource and Service

Data is typically stored in storage hardware .controlled by software. The combination of the storage hardware and this unarchitected software is the storage resource. To make the storage available in the data architecture, the storage resource is wrapped by a storage service. The storage service provides the architected interface to the storage.

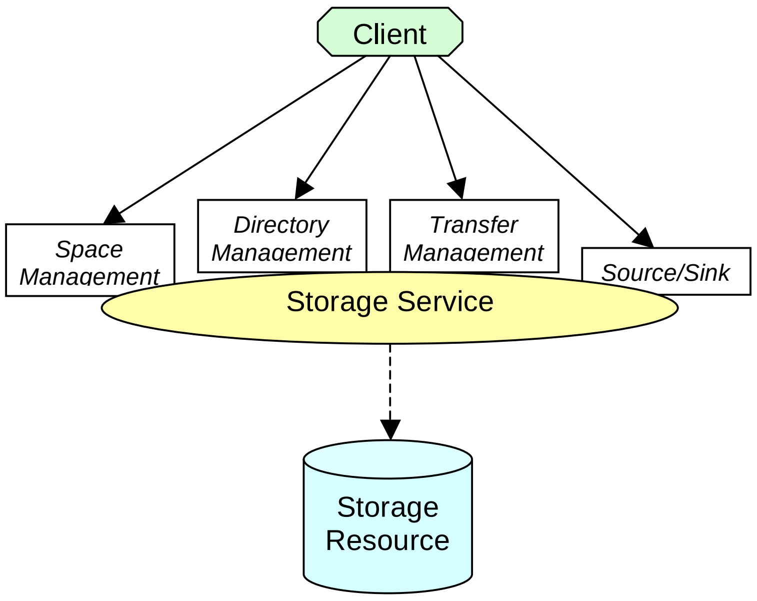

Figure 9: A storage resource wrapped by a storage resource service

The interface to data storage needs to find the commonalities of data stores while allowing the usage of specific features that may be available only on a small subset of storage resources. The three basic functional interfaces that are coupled to storage are basic data access, transfer and storage management. A storage service may additionally provide a set of optional interfaces (see below).

Data storage services on the Grid provide storage space for clients, and storage services manage storage spaces for files. A storage space has the following basic attributes (see also Storage Space Properties below for details):

- Total size.
- Allocated size.
- Unallocated size.
- Type. File storage, raw device or streaming device.
- Lifetime (in seconds say).
- Ownership and access control.
- Access properties (protocol, bandwidth, latency, etc).
- Data Retention (indicating what retention mechanism is applied to data in a given storage space).

The storage space is negotiated between a data storage service and clients using storage related policies and agreements. Each storage space created is referenced by an abstract name and an EPR

Storage is tightly coupled to all other aspects of the Data Architecture. Data access and data movement needs to take the capabilities of the storage resource into account. Storage space that is made available through storage services may be accessed through the mechanisms described in the Data Access chapter (see Section 8). Or the data may be transferred between Storage services using the Data Transfer mechanisms also described in this document (see Section 7). The management interface for Storage Resources is described in this chapter.

It is important to note that the Storage Backend that is considered here is limited to file storage – the storage of containers (files) that consist of an ordered sequence of uninterpreted bytes of data. Database storage management per se is not covered by a Grid storage resource. Databases may use file based storage or they may have their own way of managing their storage. These are private to the implementation of the database system and are not standardized. However, file space or a raw storage device provided by a Grid Storage service may well be utilized by a database to store its data. The database system may also directly use storage provided by the storage resource.

Storage services are tightly coupled to Grid scheduling. Grid schedulers need to be able to take the data requirements of a job into account: the amount of space needed for output, data needed for input, data access properties, storage QoS, etc. In order for the scheduler to be able to do the co-scheduling, the storage services need to publish their properties, perhaps into the OGSA Discovery service, and make it available through the appropriate OGSA mechanisms.

And last but not least, like any other service, storage services need to be monitored, audited and its usage has to be accountable in a standard way. Thus storage services must fit into, and comply with, the overall OGSA architecture.

9.2 Storage Service Interfaces

There are 2 key sets of Storage Management interfaces:

Space Management Operations -

> *ReserveSpace()* - facilitates negotiation of space reservation.
>
> *GetSpace()* - returns space tokens for currently allocated spaces.
>
> *ReleaseSpace()* - releases an occupied space. Depending on local settings and policies, all files may need to be released in the specified space before the space can be released.

Directory Management Operations -

> *ListFiles()* – returns a list of files and information about them.
>
> *ReleaseFiles()* - releases the files in a storage space. The files are not necessarily deleted immediately, but the space occupied by the released files is eligible for removal if space is needed.
>
> *RemoveFiles()* – removes files from a storage space.
>
> *CopyFiles()* – copies files either within a storage space or between (local) spaces.
>
> *MoveFiles()* - moves files either within a storage space or between (local) spaces.
>
> *MakeDirectory()* - creates a directory in a local storage space.
>
> *DeleteDirectory()* - deletes a directory in a local storage space.

The directory management operations provide the mechanisms for clients to manage the collection of fields held within a storage space. It is suggested that this management interface be based upon a standard interface such as that provided by the Resource Naming Service [RNS], with suitable extensions as required.

The interfaces a storage service may provide also include the basic OGSA interfaces for managing properties and negotiating agreements, and interfaces for delegation of identity. A storage service may therefore optionally provide the following management interfaces:

- Online space management interface, with agreement negotiation.
- Quota management interface, also with agreement negotiation.

- Permission (Authorization) management interface.
- User management interface.
- Request Administration Functions, such as abort, suspend and resume.
- Storage Namespace Management interface, for mapping Logical File Names to Storage URLs and Transfer URLs.

Each of these additional interfaces needs to be standardized by working groups in the OGF or some other standards body. There may also be other specific interfaces that a storage service implements and supports, which .should be discoverable through standard OGSA mechanisms.

Transfer interfaces, to initiate the transfer of data to/from the storage, are described in Section 0 of this document.

Data Access interfaces are described in Section 8 of this document.

9.3 Storage Service Properties

The properties of data storage services are listed below.

Supported Space Types – Data storage offers storage space to clients. Storage space may be requested in terms of size and type. At present three types of storage are defined:

File space: The storage space contains files. It is a valid file system with appropriate semantics. The possible types for the file system are TBD but certainly include POSIX file systems.

Raw device: The storage space is provided in raw format. It may be formatted, mounted, used as a block device, etc. The interfaces for doing this are TBD.

Streaming: The space supports only streaming mode access, reading and writing, to the data in the space. This interface needs to be defined although the ByteIO streaming interface may be appropriate.

But this set of space types is extensible and additional types could be added at a later time.

Lifetime Management – If the storage service does not support explicit lifetime management, all returned space allocations are allocations of temporary space, the default lifetime being defined by the resource properties of the storage service. If lifetime management is supported, each allocated space will have properties that define how its lifetime is managed.

Quota Support – If the storage service supports quota management agreements for space allocation then space may be refused if the requestor or the requestor's organization is running out of quota.

Online Space Management Support – Indicates whether the storage service exposes an interface to manage those files that are to be kept online in the given space. This only applies to file space types as raw and streaming spaces are defined to be always online. See below for a definition of online and nearline storage in this context.

Security Support – Indicates whether spaces available from this storage service can be requested as 'secure' spaces or not. The security semantics are governed by the corresponding OGSA security profile. For file space types this may offer say basic UNIX style security support, or a fine-grained ACL control of the files in the given space and its corresponding namespace perhaps. For raw and streaming storage the available operations to be controlled are only reading and writing.

Before even starting to negotiate an agreement for space with the given data service, these properties may be checked by clients to see whether the data service is actually capable of offering storage with the necessary semantics.

9.4 Storage Space Properties

A particular Storage Service will divide its storage resources into a number of Storage Spaces. Each storage space has a set of properties, as listed below. These are subject to negotiation through the agreement mechanism.

Size – The size of the space in bytes.

Space Type – The actual type of the given space. This can be for example '*file*', '*raw*', or '*stream*'. The *file* space type can be considered as a block of (not necessarily contiguous) file system space into which files can be copied to or retrieved from. An additional parameter for file space type is the largest single file that may be put into the given space. The *raw* space type is a block of space that can be used as a raw device, formatted, mounted as a file system, etc. The *stream* space type offers continuous streaming in or out of it (usually Write-Once Read-Many semantics).

Online – Indicating whether the space is always online (low latency) or whether it may also be nearline (i.e. on tape where the access may involve some considerable latency).

Lifetime – If the space supports lifetime management then this property indicates the time until which this space is assured to be available.

Retention Type – This property describes the behavior of the storage space when the lifetime of the data expires. Either lifetime has no effect (the data is permanent), it has the effect of the data being removed (volatile) or a notification is sent to the client so that it can take some action (durable). If the client fails to take action after a specified time out the space will be freed.

There are also many other possible properties such as Maximum Access Bandwidth, Average Access Latency, Cost, etc. Plus, whether the space is being kept on a hierarchical storage system (HSM), i.e. there may be considerable latencies in access times if the data has to be staged in from tape say before access is possible. Some of these properties may be negotiated upon space creation.

9.5 Site and VO management

Like any other service, data storage services need to take into account that they are managed both by the site owning the service and by the Virtual Organizations that have access to it. Usually sites providing storage services have the following needs and constraints while running a service:

- *Ease of provision* The site that provides a storage resource to be accessible by the Grid via a storage service wants to be able to do so with minimal effort. It should be easy to start and stop a Grid storage service.
- *VO Management*. Sites own the storage resource and service. When making available the storage service to 'the Grid', sites need to be able to sign agreements with VOs, assign quotas to VOs, restrict Quality of Service to a VO, etc.
- *Local accounting***.** The sites have to be able to charge their users for the resource usage or otherwise account for their resources, for which they usually need the information locally. It is normally sufficient to account on a per VO basis, although some sites have stricter policies.
- *Transparent user management.* A site may trust all users that are members of a given VO with which it has an agreement to provide storage. It is possible for sites to require that individual users be audited and accounted as well as individually banned from using the site resources. Some sites have policies that require everyone using their services to be personally registered (by getting an account and signing an agreement form), so this has to be possible as well if needed.
- *Local access control*. A site may have a local policy or even legal obligations to be able to track who is accessing its services and resources and they need local control to be able to control the access granted to specific users. This does not go against

the previous necessity of transparent user management – the possibility to blacklist certain users and the ability to simply track access is usually enough.

- *Quality of Service***.** The site is the defining authority for the quality of the data storage service. It is up to the site to publish the actual policies for all aspects of the storage service including data safety, data backup and availability. Virtual Organizations are responsible for the following aspects of the storage resource management and have the following constraints and needs:
- *User Management*. It is normally the responsibility of the VO to manage its users. It must be able to propagate user information to the storage service, so that users are properly authorized. The management of the actual user authorization information is the responsibility of the VO.
- *User Prioritization***.** Inside a VO some users have more rights than others. VOs need to be able to assign different priorities, quotas, abilities to each user, depending on the semantics of the actual storage service (these may be very different for a database storage service and a simple file store).
- *Data Sharing*. It is up to a VO and its users to define with whom and what other VOs the data stored in a given storage is allowed to be shared (if the data store is capable of securing the data at all).

9.6 Security Discussion

As stated above, storage spaces may support different security mechanisms. In order to enforce these, all storage interfaces to a given storage resource must be consistent in their security semantics.

The security semantics between the Data Access interfaces and the actual underlying storage space security has to be completely consistent. In some cases security can be handled by the data access facility. In other cases, it is necessary to involve the security mechanisms of the underlying storage system.

The storage services in a VO may provide many different security semantics for storage spaces (file type). The owner of a VO must carefully consider whether to allow this diversity to show to clients of the storage services of the VO or if a single set of security semantics should be chosen. The former provides a great deal of flexibility but it may lead to problems. .For instance, to provide transparent data replication, different replicas of the same file must have the same access semantics on different sites. The scenario where the very same data is accessible to a given Grid user at one site but inaccessible at another one should be avoided. However, this is unavoidable if different security semantics are used for storage within the same VO. For example, if a VO uses storage spaces at one site with POSIX ACL semantics but has a storage space with no security at another site, then all data which are stored on the insecure storage are de facto insecure at all other sites as well.

9.7 Interaction of Storage and Transfer

Storage resources play an important role in optimizing the transfers to and from a given storage space. One important special case is when a space is linked to a Hierarchical Storage Manager where all data is staged in and out to, e.g., tape or other media. In this case, it matters greatly to performance in what order the files are being accessed or are being transferred to tape.

Therefore it makes sense to define a transfer management interface for storage spaces. It can be a very simple interface that simply reorders a transfer request based on the current known state of the underlying storage. The interface has the following methods for storage spaces that support files:

OrderTransferFromStorage() – Reorders a transfer request from a storage resource based on the current known state of that resource.

OrderTransferToStorage() – Reorders a transfer request to a storage resource based on the current known state of that resource.

These take as parameters an array of File structures, each of which contains a name which identifies the file on storage. The array is then ordered and ranked. The operations return an array of RankedFile structures, each of which contains the name of the file as well as a ranking value, i.e. with what priority the file should be transferred. Lower ranking values indicate higher priority to transfer the file. Ranking values range from 0 to 10. Several files in the return list may have of course the same ranking value. The same request may return a different result at different times.

We expect that these operations will be used by the Data Transfer factory when it is creating a Transfer instance that will be moving more than one file at a time. They may also be used by applications that are directly managing the staging of files for access by those applications.

10 Cache Services

Factory Operation	Description
CreateCache	Returns an EPR to a newly created cache service.
Cache Service Operation	**Description**
SynchronizeCache	Forces (temporary) consistency of a cache.

Table 7: Cache service operations

A cache service maintains a local copy of remote data in order to improve performance. It provides the same interface to the client as the remote service and most clients can access the data without worrying about cache behavior. When the client accesses the data via the cache, the read or write is made to the cache's copy of the data. The cache maintains the consistency with the remote data, propagating updates between them, according to a specified policy. The cache service is also responsible for enforcing the security policies of the remote data.

The interface presented to the client must match that of the remote data service. It follows that there is no such thing as a generic cache service. Instead there may be many cache services, each implementing a certain set of interfaces.

The remainder of this section discusses how cache services fit into the data architecture and presents requirements on their design. These requirements need to be accepted by an as yet to be determined standards working group as input into a cache service standardization effort.

10.1 Cache Models

There are two typical uses of a cache service. In the first, the cache service is created by the remote data service to form a composite data service. The cache is purely an internal building block of the composite service, there to improve its performance. This composite service controls the cache parameters (i.e. it manages the cache). Unless the management interface of the data service provides operations that affect the cache, the clients of the composite service have no direct access to the cache at all. Thus, when a client accesses the data service, the implementation of that service may choose to access the cache to improve performance. The cache may be configured so that cache misses are automatically directed by the cache to the actual data source

In the second use, the cache service is known to the client, which may be the hosting environment, the application, a workflow enactment engine, or some other type of service. In fact, a likely scenario is that the client created the cache. The cache parameters are set by the client and the remote service will typically have no knowledge of the cache service. The cache service is deployed to increase the performance of a particular workflow, host or application. In this case the client will direct access requests to the cache service. The cache service may be configured to automatically direct cache misses to the remote data service.

In either case, the existence of the cache may be registered with the appropriate registry or discovery service. In both cases, the cache is a service and thus has its own name.

A cache service is similar in practice to a data replication service. The differences lie in intention and in the size of the data element. A cache is purely about performance. A data replication service is primarily for increasing the availability of the data or for creating snapshots for off-line processing (as in a data warehouse). In practice, data replication services may be used to enhance performance, but this is not their primary aim. Also, data replication services typically replicate well defined sets of data such as entire files or database tables. Cache services may maintain partial, dynamically changing, copies, storing just the part of the data that has been (recently) accessed by the client.

10.2 Creating Caches

A cache may be created by a factory operation, CreateCache, which is defined as follows;

CreateCache() – takes the following arguments and returns an EPR to a newly created cache service:

- The remote data source to be cached.
- The location(s) where the cache will be created. This might be a data resource or one or more storage services. The create operation may need to perform a provisioning operation at the remote location in order to create the cache.
- The data consistency requirements between the cache and the remote data.
- A description of the policy used to refresh the cache contents (if the service offers a choice).
- An indication of whether the cache is read-only or can be modified.
- An indication of the lifetime of the cache. A cache might have an indefinite lifetime or it might be created with a known time when it will be destroyed.

The result of the factory CreateCache operation is to create a cache of the specified data in the target location(s). The operation may fail, and errors will be returned, if the remote data or target location cannot be accessed or if the desired data consistency cannot be maintained.

10.3 Cache Consistency

Updates are propagated from the remote data service to the cache according to a data consistency policy. This may be predetermined by the cache service, be negotiated when the cache is created or, if the service supports it, it may be selected using a management interface.

This consistency policy has two aspects. The first is whether the cached data is up to date (unless the data is read-only). The second is the policy used to refresh the cache contents. The cache will usually only hold a subset of the remote data and the choice of refresh policy will determine which subset is held at any given time.

In general, the cache service will support a number of policies either through its management interface or agreement negotiation. These policies may allow control of cache placement, data consistency strategy, how updates directed to the cache are handled and how the contents of the cache are flushed. They may be predetermined by the cache service, be negotiated when the cache is created or, if the service supports it, it may be selected using a management interface.

There may also be times when an application needs to force (temporary) consistency of a cache. This is done via the SynchronizeCache operation:

SynchronizeCache() – takes no arguments and returns no results. It will raise a fault if the remote data source is either inaccessible or not responding.

SynchronizeCache forces synchronization of the data in the cache. This causes all writes made to the cache to be applied to the remote data source. It also either updates or clears all data in the cache. Note that by the time this operation completes, the data in the cache may already be out of date.

10.4 Cache Management

The cache service is a manageable service that can be managed by the OGSA mechanism. It must be possible to inquire about the state of the cache: what the current data consistency requirement is, what resources the cache service is consuming, what the actual current degree of data consistency is, etc. It must also be possible to control the operation of the cache service. For instance, it may be possible to change the data consistency constraint governing operation of the service. It should also be possible to control actions taken when the cache service detects errors, inconsistencies or failure to meet data consistency constraints. Other operations may be required by this document, while specific cache services may provide operations not defined here.

The following are the properties of a cache service:

- Source – the remote source of data.

- Locations – the storage space(s) used to hold the cached data.
- Data Consistency – a specification of the degree of consistency to be maintained between the remote data source and the cache.
- Refresh policy – a description of the policy used to refresh the contents of the cache.
- Read/write – an indication of whether the cache is read only or not.
- Lifetime – the lifetime of the service. A cache may have finite lifetime or indefinite lifetime.
- Security Policy – the security policy enforced on access to the cache.

11 Data Replication

Factory Operation	Description
CreateReplica	Returns an EPR to a newly created data replication service.
Replica Service Operations	**Description**
ValidateReplica	Returns an indication of whether or not the replica is identical with the primary data source.
ModifyReplicatedContents	Change the set of data being replicated.
SynchronizeReplica	Forces (temporary) data consistency of a replica.

Table 8: Data replication operations

Data items may be replicated in Grids for at least three reasons:

- fault tolerance,
- load balancing and
- to create a copy of data so that it can be processed off-line.

There are a variety of different data replication services that may be deployed in a Grid depending on the publication, discovery and data consistency requirements.

The remainder of this section discusses how replication is modeled in the data architecture and presents requirements on data replication. These requirements need to be accepted by an as yet to be determined standards working group as input into a data replication standardization effort.

11.1 Replication Modeling

Data replication has three logical components in a real world system. There is the *primary data source*. This is the actual data, managed by a data service, commonly known as “the truth”. Then there are one or more *replicas*. These are copies of some or all of the data in the primary data source. A replica will typically reflect the state of the primary data source at some point in the past. Finally there is a *data replication service* that is responsible for moving data between the primary data source and the replicas to ensure that agreed upon degrees of data consistency are maintained between the primary and replicas. The OGSA data architecture will model all three components. Each of these will be discussed later in this section.

Another model for data replication is a peer-to-peer environment in which there are no primary data sources, but, rather, all replicas are considered equivalent. In this model, a data replication service creates new replicas from existing data sources and maintains the required level of data consistency among replicas.

A replica should maintain the security policies of the source data. That is, when the data replication service creates a replica it must ensure that the proper security controls are in place (which is not to say that an administrator of the machine holding the replica could not make “inappropriate” changes to the security properties of the replica). We distinguish between managed replicas of data and copies. If an application creates a copy of the data that is not managed as a replica, then the application owns that data and can enforce whatever security it desires. Note that in a non-discretionary security environment, the system may impose restrictions on what access the copy may permit.

The discussion in the remainder of this section considers two possible scenarios for the data replication service. In the first scenario, the data replication service is only responsible for creating the replicas of data; it is independent of other functionality, such as replica discovery services. In the second scenario, the data replication service hides other logic for replica creation, discovery, etc., and offers a more opaque interface to users that handles all replica creation and access.

11.2 Creating Replicas

Replicas are created via a factory operation, CreateReplica, which is defined as follows:

> *CreateReplica()* – takes the following arguments and returns an EPR to a newly created data replication service. Creation of a replica requires specification of the following:

- Primary data source. In the case of peer to peer replication, this will be any one of the pre-existing replicas of the data.
- Target location(s) where the replica(s) will be created. This might be a data resource, a data service or a storage service. In all cases it denotes the place where the replica will be stored. Note that replication may need to perform a provisioning operation at the remote location in order to create the replica.
- SpecificationList of the data items to be replicated. This may be all or a subset of the data in the primary data source. Different types of sources may have different capabilities for defining subsets. The items to be replicated might include files, databases or subsets of those. Most, but not all, data sources will support replication as sources or targets. If a primary or target does not support data replication, an error will be returned.
- Data consistency requirements between the primary and the replica(s) (or, in the peer-to-peer case, data consistency requirements among the replicas). These will be discussed in detail in a later section. There is no guarantee than a given data consistency requirement can be met. If it cannot, an error may be returned by the creation service.
- An indication of whether the replica is read-only or can be modified.
- An indication of the lifetime of the replica. A replica might have an indefinite lifetime or it might be created with a known time when it will be destroyed.
- Should the replica be created synchronously or asynchronously?
- Replica catalog(s), if any, where the newly created replica should be registered. This may default to a well known replica registry. It is possible to declare that the replica should not be registered in a catalog. Note that the best way to name complete and partial replicas is an open issue.

The result of the replica factory CreateReplica operation is to create replicas of the specified data from the primary in the target location(s). For synchronous creation, the replica has been populated with all required data from the primary when the factory operation returns. For asynchronous creation, a handle to a replica creation service is returned to allow management of the creation process. Finally, if required, a replication process has been created to ensure that the specified data consistency between the primary and replica(s) is maintained. The operation may fail, and errors will be returned, if the primary or target cannot be accessed or if the desired data consistency cannot be maintained.

For asynchronous creation, the replica creation service will support operations to inquire about the state of an in-progress replication creation, to cancel the creation, or to modify the list of data items to be replicated or the data consistency requirements. Once creation of the replica has been achieved, these modification operations will always return an error.

Other Operations on a Replication Service

The data to be replicated may need to be changed. This is done via the ModifyReplicatedContents operation:

> *ModifyReplicatedContents()* – Change the set of data being replicated.

- Data to be removed from the replica, if any
- Data to be added to the replica, if any
- An indication of whether the newly added data should be added synchronously or asynchronously

After this operation returns successfully, the indicated data will be part of the replica. If synchronous addition has been requested then the indicated data has been initially loaded into the replica. If asynchronous addition has been specified then the data replication service may be queried to determine when the added data is available. If the data to be added cannot be added given the data consistency requirements enforced by the replica, a fault will be raised. If the data is not accessible, a fault will be raised. If removal of the indicated data results in a replica with no data in it, a warning indication will be returned.

A data replication service autonomically maintains an agreed upon degree of data consistency among the replicas and the primary data source, if any. However, there may be times when an application needs to force (temporary) data consistency of a replica. This is done via the SynchronizeReplica operation:

> *SynchronizeReplica()* – takes no arguments and returns no results. It will raise a fault if the primary data source is inaccessible or not responsive.

SynchronizeReplica forces synchronization of the data in the replica(s). There are two cases to consider:

- If there is a primary data source, updates in the replica are applied to the primary and the replica is updated with the latest data from the primary before this operation returns. Note that by the time this operation the data in the replica may already be out of date. It should also be observed that this operation does not, in and of itself, update any data in any other replicas that may exist.
- In the case of peer to peer replication, all of the replicas are updated to have the same data before this operation. In this case, the data replication service may hold all access operations until the synchronization completes. Again, upon return the data in the replica may be out of date and the various replicas temporarily inconsistent.

11.3 Managing Entries

The data replication service must provide an interface RegisterData that allows administrator clients to add and remove data from the set being replicated. This interface does not create new replicas or manage replica policy; it merely adds or removes data from previously created replication arrangements.

11.4 Discovering Replicas

Replica discovery services are used to query for the existence of replicas in a Grid environment. These services might be integrated with other general data discovery services, but they may have additional attributes that are specific to replica management.

In Naming, we discussed three types of names for data items: human-oriented names, globally unique abstract names, and low-level addresses that specify where to access a data entity. Data discovery services maintain mappings among these different names to allow discovery and access of data replicas. The discovery service must be capable of registering and searching for replicas that represent subsets of original data sources. As already mentioned, the best way to name and discover these partial replicas is an open issue.

A typical scenario for file-oriented data has two levels of discovery catalogs. The first is a *data catalog* (also known as a “metadata catalog”) that allows clients to discover the globally unique name of the data item of interest. In simple cases this might just maintains associations between human-oriented names and globally unique names. More complex examples may maintain complex descriptions of data and allow clients to query these descriptions. The second level of discovery is a *replica catalog* (also known as a “replica location service”) that maintains associations between globally unique names and physical addresses for data items. Discovery services may support the use of standard WS-names, which can be resolved to EPRs for specific data items.

- Operations to create or update mappings between globally unique names and physical addresses for data items.
- Query operations to discover the physical addresses of data items associated with a globally unique name.

- Query operations to discover the globally unique name(s) associated with a physical address of a data item.
- Operations to associate attributes with globally unique names or physical addresses of data items.
- Query operations to discover globally unique names or physical addresses of data items with specified attributes.

11.5 Validation of Registered Replicas

For many scientific applications, published data is accessed in a read-only fashion. Since replicas never change, no updates to data contents need to be propagated in the Grid. However, data items registered as replicas may become corrupted over time, either because of data transfer errors during replica creation, faults in the storage media that cause corruption after the data are stored, or because data items are modified on a remote storage system by a user or administrator in ways unknown to the publishers of the data.

For other applications that do perform data updates, these updates must be propagated to replicas by the data replication service. If errors occur during update propagation, replicas may become inconsistent with the primary data source. It is necessary to detect these inconsistencies and resolve them.

To detect replicas that become corrupted or inconsistent, it is useful to have a validation operation that can be used to check replicas and verify that they are truly replicas according to the definition of data replication imposed by the Virtual Organization. For example, the VO may require that all registered replicas have the same MD5 checksum. If the validation service discovers that a registered replica is not valid, then it can repair the replica or destroy the replica, perhaps removing it from replica catalogs.

The ValidateReplica operation performs this check on the replica:

> *ValidateReplica()* – takes no arguments and returns whether or not the replica is identical with the primary data source. If the replica is writable, this operation returns a fault.

ValidateReplica checks that the data in the replica is identical to that in the primary and returns an indication of the validity.

In some cases it might be desirable to have replicas checked on a regular and automatic basis. This can easily be done using ValidateReplica as the heart of a repetitive process defined outside of the data replication service described here.

11.6 Replica Consistency

Applications accessing replicas of data need to know the degree of data consistency between the replica(s) and the primary (or among replicas in a peer-to-peer system). The key problem is updates to the data.

For some data resources, updates to the data do not ever happen. In this case, the read only situation, data consistency is trivial – it is identity. Once the replica has been created, there is no need for any activity to maintain data consistency since the data does not change.

A second class of data resource allows the data to be updated. In this case, the degree to which the primary and target must be consistent needs to be specified. For this purpose, the data replication service must propagate updates among replicas and ensure that replica contents are valid with respect to the agreed data consistency requirement. For example, a data consistency requirement might specify that all replicas should reflect all changes that have been applied to the primary data source before a specified point in time, for example, all changes within the last ten seconds.

The data replication service is responsible for maintaining data consistency between a primary data source and one or more replicas (or among replicas in a peer-to-peer system). One possible implementation of a data replication service is to use data versioning. Updates to a data item are made at the primary data source and the data is assigned a new version number. The new version is propagated to all replica locations by the data replication service.

It is necessary to specify the allowable latency between the time that changes are received by the primary and when the updated versions of the data are available at the replica sites. Note that in this case, old versions need not be deleted, but may, instead, be deprecated in favor of later versions.

Another alternative is to have a master-slave model in which a primary replica (or the primary data source) receives any updates to a data item and the data replication service then updates the other replicas to reflect these changes. In a Grid environment, these updates are typically performed asynchronously, since synchronous updates using, for instance, a two-phase commit protocol are likely to be extremely heavy-weight and slow in the wide area network. In addition, some replicas may be unavailable for update at a given time, making a two-phase commit impossible. Asynchronous updates require the specification of data consistency constraints. In particular, it must be possible to specify the allowable latency between changes being made to the primary and those same changes being seen at the replica.

If updates are allowed at the replica(s), data consistency constraints must be specified to control the speed and manner with which these updates are propagated to the primary. Updates to the replica may be applied there and then propagated asynchronously to the primary. Updates may be applied to the primary first then to the replica before the update is acknowledged to the application. Or, the updates may be required to be applied simultaneously to the primary and the replica (e.g., via a two phase commit protocol). Finally, these options must be specified in an environment of a single primary and multiple replicas.

In a peer-to-peer environment, updates must be propagated to all the replicas. Maintaining data consistency in these environments can be challenging, since conflicting updates can be made to multiple replicas simultaneously, and the data replication service must detect and resolve these conflicts. If the system acquiesces, then the data replication service can identify conflicts and possibly force replicas into a consistent state.

11.7 Managing Replicas

As described in Replication Modeling, data replication consists of two or more end points and a data replication service that is responsible for maintaining the replicas. The data replication service is a manageable service that can be managed by the OGSA mechanisms. It must be possible to inquire about the state of the replicas: what is the current data consistency requirement, what resources is the data replication service consuming, what is the actual current degree of data consistency, etc. It must also be possible to control the operation of the data replication service. For instance, it must be possible to change the data consistency constraint governing operation of the replicas. It should be possible to control actions taken when the data replication service detects errors, inconsistencies or failure to meet data consistency constraints. These operations are examples. Others may be required by this document, while specific data replication services may provide operations not defined here.

11.8 Replication Properties

The following are the properties of a data replication service:

Source – the primary source of data. This is set to null in a peer to peer replication.

Target(s) – the replicas, the services/data resources that hold the replicated data.

Replicated data – a specification of the data being replicated.

Data consistency – a specification of the degree of data consistency to be maintained between the primary data source and the replicas. In the case of peer to peer replication this indicates the degree of data consistency among the peer replicas.

Read/write – an indication of whether the replicas are read only or not.

Lifetime – the lifetime of the replica. A replica may have finite lifetime or indefinite lifetime.

Security Policy – the security policy enforced on access to the replica(s).

Security Policy Propagation rules – an indication of how security policies should be propagated to new replicas and to the data accessed form a replica.

Replica catalog used (optional) – the set of replica catalogs that the data replication service has registered these replicas with. Note that there is no guarantee that the catalog service still has this registered data nor that the replicas have not been registered in other catalogs.

12 Data Federation

Factory Operation	Description
CreateFederation	Returns an EPR to a newly created, empty, data federation service.
Federation Service Operations	**Description**
AddSourceToFederation	Adds one or more new input sources to a data federation.
AddAccessMechanism ToFederation	Adds a new access mechanism to a dtatfederation.
UpdateFederationAttributes	Changes the various attributes of a data federation.
RemoveSourceFromFederation	Removes an input resource from a data federation.
RemoveAccessMechanism	Removes a specified access mechanism from a data federation.

Table 9: Data federation operations

In OGSA, we define data federation to be the logical integration of multiple data services or resources so that they can be accessed as if they were a single data service.

A data federation is a set of services that consists of:

- A collection of potentially distributed input data resources or data services. For instance a federation might federate a relational database, an XML database and a data service that provided a SQL interface.
- A set of filters on the data from those sources. For example, when including information from an employee database, it might be desired to exclude salary information.
- A set of transformations of that data. For example, if the federation were combining information from two geographical databases, one in France and the other in the United States, it would be necessary to do a English/Metric units conversion on one set of data or the other to ensure that distances were in the same units.
- A means to combine that data. For example, the federation might do a relational join across two of its data sources.
- A filter on the resulting data. There might be a need to exclude some of the data that results from combining the data. For example it might be necessary to exclude information about employees who live in Kansas and make more than $200,000 per year.
- A set of access mechanisms as described in 8 (Data Access), specified by URIs, that can be used to access the data provided by the federation. It is explicitly permitted for a federation to support multiple access mechanisms.
- A requested quality of service policy and a set of policies that govern the operation of the federation. A federation is responsible for managing QoS as negotiated with its clients. QoS might include specified response times, guarantees on availability, currency of data, resilience to disasters (e.g., recovery time objective, recovery point objective). Other policies might control aspects of operation such as resources consumed and security.
- An optional set of storage spaces where the federation may store either temporary data or data that represents persistent state of the data federation service.

The implementation of the federation is responsible for providing the specified access mechanisms, with the specified QoS, to its clients while accessing, filtering and transforming data from the input sources as specified. Each of these is represented by a property of the federation.

The specification of a data federation must include the means to combine input data, filters on the output data and the access mechanisms to be supported. These will define the basic way in which data from the input sources is combined and then presented to the federation client. These effectively define the data provided by the federation.

The output access mechanisms are the set of data access interfaces, as defined in the data access section of this document, that are available to clients of the federation. All access to the data provided by the federation is through one of these access mechanisms.

The policy terms are a set of policies specified using a standard OGSA policy mechanism, which represent the desired behavior of the data federation. Some of these policy terms will be defined in the specification of a data federation service while others will be specific to the particular federation being created (that is, they are defined by the service providing the federation factory operation). These terms may include quality of service (e.g., transaction rates, response times), currency of data (i.e., how stale may the data returned by, or used by, the federation may be), security, accounting and transactional characteristics. This list is intended to be suggestive rather than all inclusive. Enforcement of policies can be complicated. Thus a data federation may need the services of a sophisticated policy engine to enforce policies. This same policy engine might be used to determine if policies being requested of the federation by the various federation operations can be met.

Security and privacy are issues. A federation will access the constituent data services using normal Grid mechanisms. This means that the access will be governed by security policies. Thus the federation will need to supply security information (e.g., identity, security tokens, etc.). This may come from the security information provided by the application requesting service from the federation, it may come from the creator of the federation or it may come from an identity “owned” by the federation through some other means. Note that a federation may need to perform activities asynchronously with respect to invocations of its operations. This would seem to mean that the federation to have some identity or security information of its own.

12.1 Creation of Federations

A data federation consists of a collection of OGSA services. However, each federation is initially created empty – that is with no input sources and it exports no data. Thus the creation operation for a data federation is:

> *CreateFederation()* – takes no arguments and returns an EPR to an empty federation. It is possible that specification of the storage spaces used by the federation may be specified as part of this operation.

The client then uses the AddResourceToFederation operation to create a federation that “exports” data – see next subsection.

It is of course also possible to create a data federation “out of band” and provide an OGSA interface onto that federation as a simple service, but that is not considered further. Finally, there is the case of a data federation that combines both OGSA services and “raw” data sources (that is, sources not wrapped with an OGSA service interface). Again, this is not described as a formal part of the architecture.

12.1.1 Data Transformation

The input filters, data transformations, data combination mechanisms and the output filters represent a computational engine that takes data from the input sources and makes data available via the federation’s output access mechanisms. This computational engine performs, in general, an arbitrary computation. This computation will retrieve data from the input sources and then transform that data into the data provided by the federation. This computation may involve work required to bridge the differences in schema among the input data sources. It may be necessary to filter the input data so that not all of it used by the federation. Transformation of the input data (e.g., currency conversion) may be required prior to it being used to create the federated data. An arbitrary computation may be performed on the data that results from this schema transformation, filtering and transformation in order to produce the data provided by the federation. Finally, filtering of the resulting data from this computation may be needed before exporting the data. These computations might be done statically at the time the federation is created, or a new input source added, or they may occur dynamically when the federated data is accessed.

If the filtering and transformation of data is done statically, and not done dynamically on each access to the federation, the issue of data consistency arises. What if the data in the source data resource changes? The federation must have data consistency policies to govern how this is handled. These policies may be inherent in the federation (and thus can not be changed) or the federation may support multiple policies that can be chosen via management operations on the federation.

It is tempting to try and specify that computational engine in a declarative manner. This, however, does not appear to be technically possible, in the general case of a data federation, due to the complexity of the potential transformations that arise from the diversity of potential input sources, the large variety of schemas possible in the input sources, the difficulty of defining how data provided by these schemas can be combined (either declaratively or by relying on an automated mechanism to do the schema integration, which is an unsolved research problem)) and the diversity of potential access mechanisms that could be supported. In the specific case where the input sources are suitably constrained (e.g., all relational databases) and the access mechanism is also constrained (say, also, to be relational) then one can imagine designing a declarative specification of the transformation from input to output. Such a specification is, however, beyond the scope of this document. There will certainly be specific (classes of) federation for which a declarative specification is possible and desirable.

It is also tempting to define the operations on a federation to take an executable piece of code and embed that in some generic federation engine. Such an embedding would require the specification of the programming interface between the engine and the embedded transformation engine. The diversity of potential input and output sources makes it difficult to define suitable interfaces to make the federation work correctly and with suitable performance. Thus, we defer this to future work.

So, in the end, we are left with the need to leave the general creation of data federations outside the scope of this architecture. Federation creation will be provided by services in ways that are specific to that service.

12.2 Adding Input Sources to a Federation

The set of sources of data that a federation uses should be changeable. An application should be able to dynamically add, or delete, an input source from a federation. Similarly, an applications should be able to add or remove an access mechanism from the set of access mechanisms supported by the federation. Due to the generality of a data federation, the operations are very generic:

The AddInputSource AddResourceToFederation operation is used to add one or more new input sources to a data federation. Adding a new resource to a federation can require changes to many of the aspects of the federation. It will take the following arguments:

AddSourceToFederation() – should take the following arguments:

- The input sources to be added, specified by EPR(s). The input sources are a list of sources of data. They may be OGSA services, as specified by the OGSA naming mechanism. This operation may fail if an input source either does not exist or does not support an access mechanism required by the federation.
- The filters to be applied to the data from the new input sources.
- The transformations to be applied to the data from the new input sources.
- An update to the combination mechanisms to be used to combine all of the data from the data resources now in the federation.

- An update to the QoS requirements enforced by the federation and the policies that govern the federation's operation.
- An update to the set of access mechanisms provided by the federation.
- A specification of how the federation should incorporate the new data sources into the federation. This specification is of a form that is specific to the specific federation.

The federation will raise errors if any of the sources do not support an access mechanism needed by the federation, if any of the sources cannot be used due to some restriction imposed by the federation, if any of the sources could not be found or if there is a problem with the requested way to incorporate the new sources into the federation.

If it is desired to add a primitive source to a data federation, that must be done in a federation-specific manner and is not specified here. However, we anticipate that such operations will have much the same form as that specified here.

12.3 Other Changes to a Federation

Other changes are possible to a data federation. They are described in this section.

It is possible to add a new access mechanism to a data federation. This is done via the RemoveInputSoure AddAccessMechanismToFederation operation:

AddAccessMechanismToFederation() – adds a new access mechanism to a data federation. This may imply other needed changes to the federation so that the following arguments are required:

- The new access mechanism to be supported.
- Updates to the combination mechanisms used by the federation.
- Updates to the QoS requirements enforced by the federation.

The above operation updates to the policies that govern the operation of the data federation. Upon successful completion of this operation, the federation will support the new access mechanism. Thus, in some sense, the type of the data federation service has changed in that it now supports additional interfaces. A fault will be raised if the specified access mechanism cannot be supported or if the updates to the federation properties cannot be supported. If the operation fails, no changes are made to the access mechanisms provided by the federation

The various properties of a data federation need to be updated at times. Thus there is an UpdateFederationAttributes operation to change the items in a federation:

UpdateFederationAttributes () – Changes the various attributes of a data federation:

- Update to the filters used by the federation.
- Update to the combination mechanisms used by the federation.
- Update to the set of access mechanisms exported by the federation.
- Update to the QoS supported by the federation.
- Update to the policies that govern the operation of the federation.

Note that in some cases not all of these will need to be updated so the various arguments are all optional. UpdateFederationAttributes will raise a fault if the specified arguments are either invalid or are not supported by the federation.

Updates to policies and QoS have some special considerations. There are emerging standards for policy[4], such as [WS-Agreement] [WS-Policy], that specify how a service and its clients can determine, via an agreement interaction, the policies that govern the service's operation. For the purposes of this version of the data architecture, we only require that the federation either agree or disagree with a requested change to the QoS and policies that govern its operation. The federation is not required to participate in any agreement negotiation. As agreements and agreement negotiation are standardized, we would expect

[4] For the purposes of this discussion, it is appropriate to consider QoS as a special case of policy.

this architecture, including federations, to be updated to reflect the standardized agreement mechanisms.

Removing a set of access mechanism from an existing data federation is done via the RemoveAccessMechanism operation:

> *RemoveSourceFromFederation ()* – removes an input resource from a data federation. It takes the following arguments:

- The input source to be removed.
- Update to the filters used by the federation.
- Update to the combination mechanisms used by the federation.
- Update to the set of access mechanisms exported by the federation.
- Update to the QoS supported by the federation.
- Update to the policies that govern the operation of the federation

It takes as input the source to be removed from the federation. It will raise a fault errors if the sources are not part of the federation, if an attempt is made to remove the last input source from a federation or if removal of the source would result in a federation that cannot be supported by the federation or if the updated federation properties cannot be supported by the federation.

Removal of a primitive source from a data federation must be done in a federation-specific manner and is not specified here. However, we anticipate that such operations will have much the same form as that specified here

Adding a new access mechanism to an existing data federation is done via the AddAccessMechanism operation.

> *AddAccessMechanism ()* – adds a new access mechanism to the access mechanisms supported by the data federation. It takes the following types of arguments:

- The set of new access mechanisms to be supported.
- A federation-specific specification of how the new access mechanisms are to be supported by the federation

If for any reason the access mechanism(s) cannot be supported, the federation will raise an error. Upon successful completion of this operation, the federation will now support the new access mechanisms. Thus, in some sense, the type of the federation service has changed in that it now supports additional interfaces. If the operation fails, no changes are made to the access mechanisms provided by the federation.

Removing a set of access mechanism from an existing data federation is done via the RemoveAccessMechanism operation:

> *RemoveAccessMechanism ()* – removes a specified access mechanism from a data federation. This may require changes to some of the federation properties. This takes the following arguments:

- The access mechanism to be removed
- Update to the filters used by the federation.
- Update to the combination mechanisms used by the federation.
- Update to the set of access mechanisms exported by the federation.
- Update to the QoS supported by the federation.
- Update to the policies that govern the operation of the federation.

A fault will be raised if an attempt is made to remove the last access mechanism from a data federation or if the specified changes to the federation's properties cannot be supported given the set of access mechanisms that would result after removal of the specified access mechanism. Upon successful completion of this operation, the federation will no longer support access via the indicated access mechanisms. Applications that use this federation via one of the removed access mechanisms will now fail with a suitable error raised from the service infrastructure. If a fault is raised, no changes are made to the federation.

12.4 Access to the Federation's Data

The data provided by the data federation is available to applications through the set of access mechanisms supported by the federation. These access mechanisms are specified elsewhere in this document.

A federation may choose to support partial results. That is if satisfying an access request requires data from one or more input resources that are not currently available, the federation may choose, if the policies governing federation operation permit it, to return a partial result. In this case the federation must also return an indication to the requestor that partial results have been returned as well as an indication of which resources were not available. The requestor should have the ability to request, either as part of the access operation or by changing the federation's governing policies, that no partial results be returned. Note that supporting partial answers may require extensions to standardized access mechanisms. Such extensions are beyond the scope of this document. To support optimized data access, including the important case of access via a query, we expect that the basic access mechanisms provided by the input sources will be extended with additional operations to facilitate this optimization. For instance, a federation that takes relational sources as input might desire those sources to provide operations to describe available indices, sizes of tables and estimations of the size of result sets. A federation that supports transactional semantics might require facilities from its input sources that support distributed, two phase commit.

12.5 Updates to Policy

As with all services, a data federation will provide operations to replace and modify the policies governing its operation. In considering the implications of these operations on a service, it is important to note that the ability of a federation to follow a policy may be dependent upon the capabilities of the services and resources that comprise the federation. In some cases the federation may be able to take actions to follow a policy in spite of the limitations of its constituent services and resources. In other cases the federation will not be able to take such compensating actions and it will reject such policies.

12.6 Access to the State of the Federation

Clients may need to have access to parts of the state of a data federation. This section enumerates those properties that are available. The actual mechanism for accessing this state is specified in an OGSA profile.

The properties of the accessible state of a federation, as described earlier, are accessible via standard OGSA mechanisms:

- Input sources: the set of input data sources for the federation.
- Access mechanisms: the set of access mechanisms currently supported by the federation.
- Policies: the set of policies governing the operation of the federation.

A federation might choose to create replicas of data and cache data from one or more of its input sources. A federation may choose to make the existence of these known to its clients either as state provided by the federation or by registering a replica or cache with an appropriate registry.

12.7 Security Considerations

A data federation must take suitable measures to ensure the security and privacy of the data that it provides. Being a Grid service, the implementation of a data federation has its operation limited by security and privacy considerations. The generality and complexity of a federated environment, and the many potential federation patterns, makes it impossible to explicitly discuss all possible cases. Thus this section will outline general principles that a data federation must adhere to.

A data federation will access the constituent data services using either normal Grid mechanisms or via some native, non-Grid, access mechanism. In either case this means that the access will be governed by security policies. Thus the federation will need to supply security information (e.g., identity, security credentials, etc.) to the data source that it is accessing. This security information may come from one of several sources:

- The security information may be provided by the application requesting service from the federation.
- The identity of the requesting application might be used directly.
- The federation might have its own identity and use that to access the data sources. This identity might have been supplied when the federation was created or it may have been provided as a (changeable) property of the federation.

In any of these cases, this security information will govern what data the federation can access.

Note that a data federation will frequently perform activities asynchronously with invocations of its operations. This would seem to imply that the federation must have some inherent identity or security information in order to perform those operations when they involve access to either other Grid services or to its input data sources.

The data federation is responsible for attaching security and privacy controls to the data that it returns, using the normal Grid security and privacy mechanisms discussed in Section 4. The nature of these controls (e.g., the access control lists for the data) may come from one or more of multiple sources. The federation may have one or more properties that, in union, define the access allowed to the data coming out of the federation. The federation may attach a security policy to the exported data that is a function of the access controls on the data extracted from its data sources. Clearly the data federation cannot, in and of itself, enforce these attached security controls. This is the responsibility of the security mechanisms being used.

12.8 Standardization Considerations

The description above is of great generality. At this time it appears as if the work required to standardize data federations has considerable research content. Thus we recommend formation of an OGF research group to explore data federation. This working group would be responsible for identifying those parts of data federation that are ready for standardization as well as attempting to resolve the remaining issue. This research group should also attempt to identify specializations of data federations that are of high value to the community and which are well enough understand to be amenable to standardization. For instance, one quite likely specialized data federation is one that takes multiple relational sources as input and supports relational access to the federation. Such a specialization can build upon the work of research projects such as OGSA-DAI and existing products such as IBM’s WebSphere Information Integrator.

13 Data Catalogs & Registries

Registry Operations	Description
Publish	Add an entry to the catalog.
Update	Modify an existing entry.
Augment	Add additional properties for an entry created by someone else.
AddClassification	Add a classification to the catalog.
Classify	Specify how descriptions are classified according to registered classifications.
Find	Apply a query to the catalog and return matching entries.

Table 10: Registry operations

A *data catalog* is a service that provides mechanisms for publishing and querying descriptions of data services, their capabilities and the content they represent. Data catalogs are a particular case of registries, as in principle a registry can store descriptions of arbitrary items. In this section we describe the functionality required for data services, which need to be accepted by an as yet to be determined standards working group as input into a standardization effort. This hypothetical group may specify a standard for data catalogs in particular; alternatively it may develops a standard for registries, with data catalogs as simply a particular use case of that standard.

Data catalogs are often called “metadata catalogs”. In keeping with our principle of avoiding the overloaded term “metadata”, we choose our term by the fact that the catalog contains descriptions of data (much as a clothing catalog contains descriptions of clothing or a component catalog contains descriptions of components).

13.1 A model for data catalogs

The descriptions stored in a data catalog for a particular service may originate from the data service itself or from third parties. For example, the service may choose to register itself with a registry so that clients may discover that service. Alternatively, the registry may be populated by an automatic discovery mechanism such as a gridbot. Third parties may also store information about services such as assessments of quality or relationships with other services. In both cases the descriptions are written in XML.

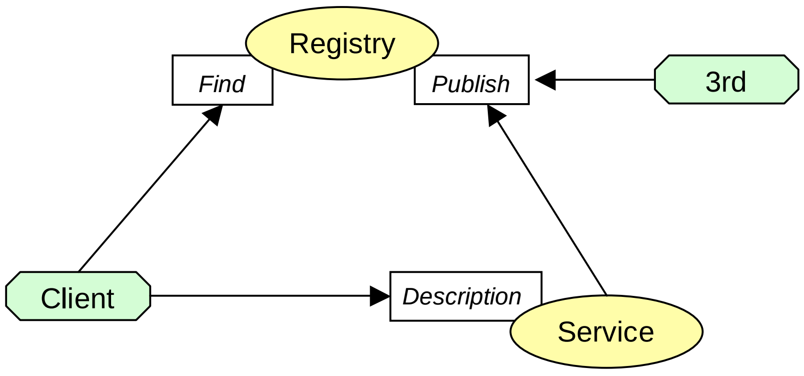

Figure 10: A model for data catalogs

To satisfy the requirements of such a service, a registry should provide operations for:

- Managing the data descriptions.
- Publication and removal of data descriptions.
- Discovering services and resources and obtaining descriptions relating to them.

- Annotation of existing data descriptions.
- Subscribing for notification of changes in data descriptions.

The registry may aid service discovery by providing for the standard categorization of the descriptions that it holds.

13.2 Publication

It should be possible to publish metadata securely, where the publisher retains ownership of the description and is the only person that can modify it or anonymously, where the description has no owner. This is done via a Publish operation:

Publish () – Add an entry to a catalog.

- Name identifying the entity to be described.
- The description of that entity to be added.
- Security policy.
- Currency policy.

The result of the Publish() operation will be success or failure. Lifetime management functionality may be provided by associating EPRs with content and relying on the OGSA Basic Profile.

It must also be possible for the owner of a description to update it. This is done via the Update operation:

Update () – Modify an existing entry.

- Name identifying the entity to be described.
- The new description of that entity.

A registry may also allow a third party to add to the description of an existing entry, even if they do not have the ability to modify the original description. The query mechanism may provide facilities whereby queries can choose to take account or ignore descriptions from third parties. The addition of third-party descriptions is provided by the Augment operation:

Augment () – Add additional properties for an entry created by someone else.

- Name identifying the entity to be described.
- The additional description of that entity.

13.3 Classification

Some registry systems, such as UDDI [UDDI], provide mechanisms for classifying descriptions. This allows a registry manager to upload a categorization scheme and then assign entries in the registry to these categories. The intention is that clients can then find content by category. This is supported by the AddClassification and Classify operation:

AddClassification () – Add a classification to the catalog.

- The name of the classification scheme.
- An XML specification of the tags in the classification scheme.

AddClassification() does not return a result, unless faults occur.

Classify () – Specify how descriptions are classified according to registered classifications.

- A list of (scheme name, tag) pairs.

Classify() is called on a registered description to add that description to the appropriate classification. There probably needs to be an additional operation that does this for a list of descriptions, rather than having to call the operation once for each description.

The increasingly popular use of tagging is a less formal approach to categorization where individuals just invent words to indicate categories and then associate them with content.

13.4 Query

The fundamental client operation on a data catalog is to query the descriptions to find items of interest. This is done with the Find() operation:

Find () – Apply a query to the catalog and return matching entries.

- Query expression.
- The new description of that entity.

Information provided by a registry is exposed in XML format and is queried using XPath or XQuery statements. The result of this operation will be references to services that satisfy the query.

If the result of service or resource discovery through the registry is an address it can be used directly to locate the service providing access to the data resource. When the result comprises a service or resource name further work must be undertaken to retrieve the address associated with the returned name, possibly using another registry.

13.5 Currency

When descriptions are held in a registry, the registry is effectively acting as a cache. Therefore the information should be refreshed using selectable strategies. For example,

- Do not cache, i.e. always read directly from the service.
- Cache once, i.e. the data is static and will not change.
- Cache timeout, i.e. it is not vital that the data is absolutely up to date so refresh every n seconds.
- Update on event, i.e. update the cache whenever the data changes.
- Update on age, i.e. give the information an age and provide policy to drive updates to aging data.

The precise set of policies that govern the operation of a data catalog, including these currency properties, needs to be defined. In addition there will also be properties that describe the registry itself – e.g. the age of a particular entry, capabilities, etc.

It should be possible for a client to subscribe to any changes in the description of a given entity. This functionality should be provided by the notification features of an OGSA Basic Profile and possibly the work of the Info-D Working Group of OGF [Info-D].

13.6 Security and Hierarchies

Registries may employ a variety of trust models. In order for a service to publish descriptions with a registry it may have to pre-register with the registry. This reassures a client that the description comes from a trusted source. The content that may be queried may depend on the credentials used by a client. The trust model used within a Virtual Organization may be implicit, i.e. members of that VO are automatically trusted. However as annotations may be replicated from one registry to another as in the figure below, other VO domains may be less trusted and the amount of information replicated may be a subset of that available at the original registry. A registry client may have to drill down through the services in order to obtain the full information.

Registries that span VO boundaries are out of scope for this version of this document.

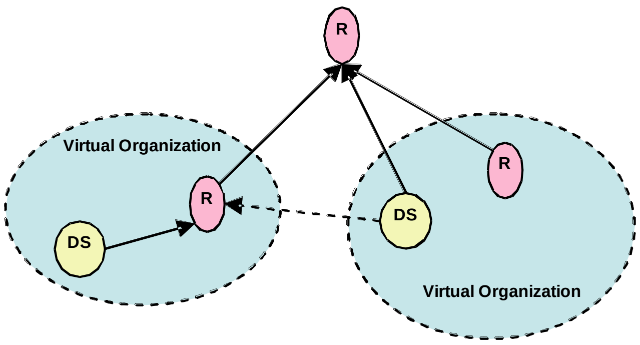

Figure 11: Registries that span multiple Virtual Organizations

14 Conclusions and Future Work

This OGSA Data Architecture is the first attempt to define the data management capability within the OGSA architecture. It addresses the issues that pertain to data within Grids and the services, operations and properties required to deal with data in a manner consistent with the OGSA vision. The document is not supposed to be prescriptive but rather highlight and suggest what needs to be present in Grid environments to deal with data. No doubt, as more detailed examinations are made of the requirements, services are implemented and further investigations are carried out, the details contained within this document will have to change. However, this document provides a base-line from which such work may start.

The OGSA Data Working Group produced this architecture with the twin intents of:

- linking existing work on standards for data grids with the ongoing development of the OGSA architecture.
- stimulating new standardisation work in key areas.

This work has achieved these goals not only by producing the current document but also by:

- contributing the description of data management capabilities to the OGSA 1.5 document
- creating the successful OGSA ByteIO working group in OGF
- encouraging the OGF's Grid File Systems working group to develop RNS to meet the needs of data grids
- stimulating the creation of the OGSA WS-Naming profile document
- launching the OGSA DMI working group and continuing to influence its work

Version 1.0 of the OGSA Data Architecture covers only some basic capabilities. Future versions should tackle key ideas such as session management, integration with workflow systems, integration with provisioning systems and operations on streams. There is much that needs to be done on infrastructure for managing vocbularies and ontologies. We believe that version 1.0 will form a sound base on which these futher developments can be built.

15 Appendix: Summary of General Interfaces

This appendix gives a summary of all the interfaces defined in this document.

15.1 Policy Operations

Operations	**Description**
ReplacePolicy	Replaces the entire policy governing the operation of the service with the policy supplied as an argument.
UpdatePolicy	Updates part of the policy governing operation of the service

15.2 Data Transfer Operations

Factory Operation	**Description**
CreateTransfer	Initiates a transfer between a data source and a data sink. Returns an EPR to a Transfer service that is managing the transfer.
Data Resource Operations	**Description**
GetHandle	Returns a WS-Name for the transfer related operations provided by that service
GetSupportedProtocols	Returns the set of transfer protocols supported by this data resource.
SetupTransfer	Returns a protocol-specific URI which can be used to initiate and mange the transfer.
Transfer Service Operations	**Description**
PauseTransfer	Temporarily stops an in-progress transfer.
ResumeTransfer	Resumes a paused transfer.
StopTransfer	Stops an in–progress transfer. The Transfer service is destroyed as part of this operation.

15.3 Data Access Operations

Factory Operation	**Description**
Create	Create an association between a data service and an underlying resource with a given service interface. The underlying resource may be created and populated as a result of this operation.
Unstructured Data Access Operations	**Description**
Read	Takes an offset and a number of bytes to read, and returns that number of bytes from the resource.
Write	Takes a fixed number of bytes of data which it writes to the resource.

Structured Data Access Operations	Description
ExecuteQuery	Provide a means for querying a resource. This could take an SQL, an XPath or XQuery expression or any language type described by the LanguageMap property which is then run by the underlying resource.
BulkLoad	Provide a means of inserting large amounts of data into an underlying resource.

15.4 Storage Operations

Space Management Operations	Description
ReserveSpace	Facilitates negotiation of space reservation.
GetSpace	Returns space tokens for currently allocated spaces.
ReleaseSpace	Releases an occupied space. Depending on local settings and policies, all files may need to be released in the specified space before the space can be released.
Directory Management Operations	**Description**
ListFiles	Returns a list of files and information about them.
ReleaseFiles	Releases the files in a storage space. The files are not necessarily deleted immediately, but the space occupied by the released files is eligible for removal if space is needed.
RemoveFiles	Removes files from a storage space.
CopyFiles	Copies files, either within a storage space or between (local) spaces.
MoveFiles	Moves files either within a storage space or between (local) spaces.
MakeDirectory	Creates a directory in a local storage space.
DeleteDirectory	Deletes a directory in a local storage space.
Transfer Management Operations	**Description**
OrderTransferFromStorage	Reorders a transfer request from a storage resource based on the current known state of that resource.
OrderTransferToStorage	Reorders a transfer request to a storage resource based on the current known state of that resource.

15.5 Cache Service Operations

Factory Operation	Description
CreateCache	Returns an EPR to a newly created cache service.
Cache Service Operation	**Description**
SynchronizeCache	Forces (temporary) consistency of a cache.

15.6 Replication Operations

Factory Operation	Description
CreateReplica	Returns an EPR to a newly created data replication service.
Replica Service Operations	**Description**
ValidateReplica	Returns an indication of whether or not the replica is identical with the primary data source.
ModifyReplicatedContents	Change the set of data being replicated.
SynchronizeReplica	Forces (temporary) data consistency of a replica.

15.7 Federation Operations

Factory Operation	Description
CreateFederation	Returns an EPR to a newly created, empty, data federation service.
Federation Service Operations	**Description**
AddSourceToFederation	Adds one or more new input sources to a data federation.
AddAccessMechanism ToFederation	Adds a new access mechanism to a data federation.
UpdateFederationAttributes	Changes the various attributes of a data federation.
RemoveSourceFromFederation	Removes an input resource from a data federation.
RemoveAccessMechanism	Removes a specified access mechanism from a data federation.

15.8 Registry Operations

Operations	Description
Publish	Add an entry to the catalog.
Update	Modify an existing entry.
Augment	Add additional properties for an entry created by someone else.
AddClassification	Add a classification to the catalog.
Classify	Specify how descriptions are classified according to registered classifications.
Find	Apply a query to the catalog and return matching entries.

16 Appendix: Interface Mappings to Existing Specifications

This section maps the proposed interfaces in the OGSA Data Architecture to interfaces proposed in existing standards.

16.1 Data Access Mappings

This subsection describes the mapping of the Data Access Section to existing OGF standards. There are two main specifications in this area – those from the OGF DAIS WG for structured data and from the OGF OGSA ByteIO WG for unstructured data.

The WS-DAI specification [WS-DAI] produced by the *Database Access and Integration Services* Working Group, DAIS-WG, proposes a generic, model independent way of providing access to structured data held in data resources. The WS-DAI specification provides a set of basic patterns that are then specialized for particular types of data resources, such as relational databases or XML databases, in related specification documents referred to as realizations.

Operation	**WS-DAI mapping**
Create	Message pattern defined but no operations are defined.
ExecuteQuery	CoreDataAccess::GenericQuery
BulkLoad	<None>

DAIS realizations seeks to extend the core properties and operations defined in the core specification for a particular type of data model: this is done by [WS-DAIR] for relational data resources and the [WS-DAIX] specification for XML data resources. This ensures that different realizations have a level of functional commonality.

Operation	**WS-DAIR mapping**
Create	SQLAccessFactory::SQLExecuteFactory SQLResponseFactory::SQLRowsetFactory
ExecuteQuery	SQLAccess::SQLExecute
BulkLoad	<None>

Operation	**WS-DAIX mapping**
Create	XMLCollectionFactory::CollectionSelectionFactory XQueryFactory::XQueryExecuteFactory XPathFactory::XPathQueryFactory
ExecuteQuery	XQueryAccess::XQueryExecute XPathAccess::XPathExecute XUpdateAccess::XUpdateExecute
BulkLoad	<None>

The ByteIO specification [OGSA ByteIO] is divided into two separate and distinct interfaces – RandomByteIO and StreamableByteIO – each addressing a unique set of use cases. The first of these supports the notion that a data resource is directly accessible and that clients can handle the maintenance of any session state (such as size and position). This is specifically designed to ease the burden of service authoring by pushing much of the management to the client libraries. The other interface is useful for clients that wish for more session-like semantics in their data interactions. In this latter case, resources implementing the interface do not represent the data source/sink directly so much as an open session between the client and the data. This distinction is most clear when considering the canonical semantics of "terminating" resources. When a RandomByteIO resource is terminated, the data resource is destroyed but when a StreamableByteIO resource is terminated, the data is unaffected -- only the open session between the client and the data disappears.

Operation	RandomByteIO mapping	StreamablebyteIO mapping
Read	RandomByteIO::read	StreamableByteIO:: seekRead
Write	RandomByteIO::write RandomByteIO::append RandomByteIO::truncAppend	StreamableByteIO:: seekWrite

16.2 Storage Management Mappings

This subsection describes the mapping of the Storage Management section to the existing OGF specification SRM [SRM].

Space Management Operation	SRM mapping
ReserveSpace	srmReserveSpace
GetSpace	srmGetSpaceTokens
ReleaseSpace	srmReleaseSpace

Directory Management Operation	SRM mapping
ListFiles	srmLs
ReleaseFiles	srmRm
RemoveFiles	srmRm
CopyFiles	srmCopy
MoveFiles	srmMv
MakeDirectory	srmMakeDir
DeleteDirectory	srmRmDir

Transfer Management Operation	SRM mapping
OrderTransferFromStorage	<None>
OrderTransferToStorage	<None>

17 Appendix: Specifications referred to in this document

The role of this document is to explain how relevant specifications can fit together to create a coherent Grid architecture for data based on web services. This section briefly lists the relevant specifications. Each of these specifications describes a different part of this architecture. Some have much wider applicability; for example the descriptions of data formats are used in a wide variety of systems.

Some of these specifications already exist, others are in development.

17.1 Data Access Specifications

Specification	**Description**	**Defining body**
ByteIO	POSIX-like operations for Grids.	OGF (OGSA ByteIO WG)
WS-DAI	Virtualization of queries over structured data	OGF (DAIS WG)
WS-DAIR	Instantiation of WS-DAI for relational data	OGF (DAIS WG)
WS-DAIX	Instantiation of WS-DAI for XML data	OGF (DAIS WG)

17.2 Data Description Specifications

Specification	**Description**	**Defining body**
DFDL	Describing binary data	OGF (DFDL WG) – under development
Dublin Core	Common metadata elements	DCMI
JPEG	Image format	ITU/CCITT
PNG	Portable Network Graphics	IETF
TIFF	Image format	Adobe
XSD	XML schema description	W3C

17.3 Data Transfer Specifications

Specification	**Description**	**Defining body**
FTP	File Transfer Protocol	IETF
GridFTP	Extended FTP	OGF (GridFTP WG)
MTOM	SOAP-based binary data	W3C
OGSA DMI	Data Transfer Control	OGF (OGSA DMI WG) – under development
SwA	SOAP with attachments	W3C
DIME	SOAP-based binary data	IETF – withdrawn

17.4 Storage Specifications

Specification	Description	Defining body
SRM	Management of mass storage resources	OGF (GSM WG)

17.5 Infrastructure Specifications

Specification	Description	Defining body
Info-D	Publish-subscribe information dissemination	OGF (InfoD WG)
JSDL	Job submission description language	OGF (JSDL WG)
OGSA Basic Security Profile	Messaging security	OGF (OGSA WG)
OGSA WSRF Basic Profile	Resource addressing, Lifetime management, Notification	OGF (OGSA WG)
RNS	Human-meaningful names	OGF (GFS WG)
UDDI	Universal Description Discovery and Integration	OASIS
URI	Universal Resource locator	W3C
WS-Addressing-core	Transport-neutral addresses	W3C
WS-Agreement	Policy agreement management	OGF GRAAP WG
WS-DM	Management	OASIS
WS-Management	Management	DMTF
WS-Naming	Abstract names	OGF OGSA-Naming WG
WS-Policy	A framework for policy definition	W3C

17.6 Transaction Specifications

Specification	Description	Defining body
WS-Coordination, WS-Atomic Transaction and WS-BusinessActivity	Distributed transactions	OASIS
WS-Context, WS-Coordination Framework, WS-Transaction Management	Distributed transactions	OASIS

17.7 API Specifications

Specification	Description	Defining body
POSIX	Byte stream IO	IEEE
NFS	Network file access	IETF
CIFS	Network file access	SNIA
JDBC	Java database access	Sun
ODBC	Database access	Microsoft

18 Glossary

This section defines the terms used in this document and gives the origin for these definitions. Although we have tried to use terms in their generally accepted manner, this glossary is the definitive definition of the terms used in this document, along with the OGSA glossary [OGSA Glossary].

Term	Definition	Origin
API	Application Programming Interface – a set of language-specific types, methods and other features that allow an application to access or control a resource.	New
Capability	In OGSA, a set of one or more services that together provide a function that is useful in a Grid context. The OGSA Data Management Services described in this document are an example of an OGSA capability.	OGSA Glossary v1.5 (definition) New (example)
Cache	A cache service maintains a local copy of a subset of remote data, in order to improve access times for local consumers.	New
Component	An interchangeable part of a system that encapsulates its contents and defines its behavior in terms of its public interfaces.	OGSA Glossary v1.5
Data Access	A mechanism that allows an entity to identify a subset of the data held by a data resource and either update that subset, return it to the requesting entity, or make it available for transfer elsewhere.	New
Data Catalog	A *registry* that stores *data descriptions* of *data services* or the *data resources* they represent. . This is sometimes called a “metadata catalog”.	
Data Consistency	A policy that specifies or describes how up-to-date an instance of data is in one service of a distributed system, with respect to one or more instances of that data elsewhere in the system.	New
Data Description	A generic term for the set(s) of properties that specify aspects of particular data, including the *data format*, information about the resource that holds the data, and any other description of that data such as provenance, classification, textual description, etc.	New
Data Federation	In OGSA, *data federation* refers to the logical integration of multiple *data services* or *data resources* so that they can be accessed as if they were a single service.	OGSA Glossary v1.6
Data Format	The encoding, structure, classification and organization of data in a *data resource* or message.	New
Data Management Services	In OGSA, the *capability* concerned with the storage, description, access, update, location, transfer and other management of data.	New
Data Replication	The maintenance of one or more copies (replicas) of data such that the replicas are kept up to date with any changes in that data, to a degree specified by an applicable *consistency* policy.	New

Term	**Definition**	**Origin**
Data Resource	An entity (and its associated framework) that provides a *data access* mechanism or that can act as a *data source* or *data sink*.	New
Data Service	A *service* that provides *interfaces* to the *capabilities* and data of one or more *data resources* within a *service-oriented architecture*.	New
Data Set	An encoding of data in a defined syntax suitable for externalization outside of a *data resource*, for example for *data transfer* to or from another *data resource*. Examples include a WebRowSet encoding of an SQL query result set, a JPEG-encoded byte array, and a ZIP-encoded byte array of a set of files.	Derived from OGSA Data Services, Dublin Core Metadata Initiative [Dublin Core]
Data Sink	A *data resource* that receives the data copied by a *data transfer* mechanism from a *data source*.	New
Data Source	A *data resource* that contains the data to be copied to a *data sink* via a *data transfer* mechanism.	New
Data Staging	The transfer of data to a specified location in preparation for an activity, e.g. running a job on an execution resource, or the transfer of data resulting from an activity to another location.	New
Data Transfer	A mechanism to physically copy data from a *source* to a *sink*.	New
Entity	Any nameable thing. For example, in *OGSA* an entity might be a *resource* or a *service*.	OGSA Glossary v1.5
Interface	In a service-oriented architecture, a specification of the operations that a service offers its clients. In *WSDL* 2.0 an interface component describes sequences of messages that a *service* sends and/or receives.	OGSA Glossary v1.5 New
Management	The process of taking administrative actions such as *deploying*, configuring, monitoring, metering, tuning, and/or troubleshooting *resources*, either manually or automatically.	OGSA Glossary v1.6
Metadata	Data that describes data. Metadata may include references to schemas, provenance, and information quality.	OGSA Glossary v1.5
Quality of service (QoS)	A measure of the level of service attained, such as security, network bandwidth, average response time or service availability.	OGSA Glossary v1.5
Replica Catalog	A registry that maintains associations between globally unique names and physical addresses for replicated data items	New
Registry	An authoritative, centrally-controlled store of information. *Web services* use registries to advertise their existence and to describe their *interfaces* and other attributes. Prospective *clients* query registries to locate required *services* and to discover their attributes.	OGSA Glossary v1.5

Term	**Definition**	**Origin**
Resource	A resource is a physical or logical *entity* that supports use or operation of a computing application or environment. In a Grid context the term encompasses entities that provide a capability or capacity (e.g., servers, networks, disks, memory, applications, databases, IP addresses, and software licenses). Dynamic entities such as processes, print jobs, database query results and *virtual organizations* may also be represented and handled as resources. See http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#resource for the WS Architecture definition of this term.	OGSA Glossary v1.6
Scenario	A scenario is a specific sequence or path of interactions, from initiation to goal, occurring within a particular environment and/or *context*. A *use case* may contain multiple scenarios. OGSA scenarios are high-level and described in a casual style.	OGSA Glossary v1.5
Service	A *service* in the most general sense is an entity, usually composed of one or more software *components*, that provides functionality in response to client requests. A service is often a part of a *service-oriented architecture*, and participates in realizing one or more *capabilities*. For example, an electronic bookstore application is a service, and its database component provides a database service to the bookstore. Thus high-level services may be decomposed into lower-level constituent services. In general, a service and all of its decomposable sub-services are considered to be *components*.	OGSA Glossary v1.6
Service-oriented architecture (SOA)	This term is increasingly used to refer to an architectural style of building reliable distributed systems that deliver functionality as *services*, with the additional emphasis on loose coupling between interacting services. Note: An SOA can be based on *Web services* (which provide basic interoperability), but it may use other technologies instead.	OGSA Glossary v1.5
Storage Resource	A resource that provides a physical or logical storage capability.. Examples include storage devices, storage appliances, disk volumes and file systems.	New
Virtualization	Virtualization uses a level of indirection to abstract the implementation details of one or more entities, enabling them to appear to their consumers in a more appropriate form. For example, a virtualized entity might present different interfaces from its underlying entities, a single entity might be partitioned and presented as a set of (lower-capacity) entities, or a set of discrete entities might be treated as a single aggregate entity.	OGSA Glossary v1.6

Term	**Definition**	**Origin**
Virtual organization (VO)	A virtual organization comprises a set of individuals and/or institutions having direct access to computers, software, data, and other *resources* for collaborative problem-solving or other purposes. VOs are a concept that supplies a *context* for operation of the *Grid* that can be used to associate users, their requests, and a set of resources. The sharing of resources in a VO is necessarily highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs.	OGSA Glossary v1.5

19 Contributors

19.1 Author Information

Dave Berry
National e-Science Centre
15 South College Street
Edinburgh
EH10 4ST
UK

Allen Luniewski
IBM Cross Brand Services
IBM Silicon Valley Laboratory
555 Bailey Ave.
San Jose, CA 95141

Mario Antonioletti
EPCC
JCMB
The King's Buildings
Mayfield Road
Edinburgh
EH9 3JZ
UK

19.2 Acknowledgements

This document contains contributions from many people who have given their time and expertise towards the production of the OGSA Data Architecture. The authors thank the people who have contributed text, ideas and comments to this document, including the following:

Ted Anderson, IBM Corporation, USA

Malcolm Atkinson, NeSC, UK

Ann Chervenak, Information Sciences Institute, USA

Neil Chue Hong, EPCC, UK

Stephen Davey, NeSC, UK

Abdeslem Djaoui, CCLRC, UK

Andrew Grimshaw, University of Virginia, USA

Shannon Hastings, Ohio State University, USA

Peter Kunszt, CERN, Switzerland,

Simon Laws, IBM Corporation, UK

Leo Luan, IBM Corporation, USA

Fred Maciel, Hitachi, Japan

Susan Malaika, IBM Corporation, USA

Mark Morgan, University of Virginia, USA

Manuel Pereira, IBM Corporation, USA

20 Intellectual Property Statement

21 Disclaimer

22 Full Copyright Notice

23 References

[CIFS]
Storage Network Industry Association. *Common Internet File System (CIFS)* Technical Reference. Revision 1.0. http://www.snia.org/tech_activities/CIFS/CIFS-TR-1p00_FINAL.pdf.

[DAIS]
Atkinson, M,. Dialani, D., Guy, L., Narang, I., Paton, N., Pearson, D., Storey, T., Watson, P., *Grid Database Access and Integration: Requirements and Functionalities*, Global Grid Forum, Lemont, Illinois, U.S.A., GFD-I.13, March 2003. http://www.ogf.org/documents/GFD.13.pdf.

[DIME]
Nielsen, H. and Sanders, H.. Direct Internet Message Encapsulation (DIME). Microsoft, June 2002. http://xml.coverpages.org/draft-nielsen-dime-02.txt

[Dublin Core]
DCMI Usage Board, *DCMI Metadata Terms*, DCMI Recommendation, December 2006. http://dublincore.org/documents/dcmi-terms/.

[FTP]
Postel, J. and Reynolds, J. *File Transfer Protocol (FTP).* IETF RFC 989, October 1985. http://www.ietf.org/rfc/rfc0959.txt

[GridFTP]
Mandrichenko, I., Allcock, W. and Perelmutov, T. *GridFTP v2 Protocol Description.* OGF GFD.47, May 2005. http://www.ogf.org/documents/GFD.47.pdf

[Info-D]
Davey, S., Dialani, V., Fehling, R., Fisher,S., Gawlick, D., Kantarjiev, C. Madsen, C.l Malaika, S.l Mishra, S. and Shankar. M., *Information Dissemination in the Grid Environment – Base Specifications*, GFD-R-P-110, Open Grid Forum, July 2007. http://www.ogf.org/documents/GFD.110.pdf.

[JDBC]
Andersen, L., Specification Lead. *JDBC™ 4.0 Specification, JSR221*. Sun Microsystems, Inc. November 7, 2006.

[JPEG]
Information Technology Digital Compression and Coding of Continuous-Tone Still Images Requirements and Guidelines,. JPEG ISO/IEC 10918-1 ITU-T Recommendation T.81, 1994.

[JSDL]
Anjamshoaa, A., Brusard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D. and Savva, A., *Job Submission Description Language (JSDL) Specification, Version 1.0*. GFD-R.056, Global Grid Forum, November 2005. http://www.ogf.org/documents/GFD.56.pdf.

[MTOM]
Gudgin, M., Mendelsohn, N., Nottingham, M. and Rujellan, H. *SOAP Message Transmission Optimization Mechanism*. W3C Recommendation, January 2005. http://www.w3.org/TR/soap12-mtom/.

[NFS]
Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M. and Noveck, D. *Network File System (NFS) version 4 Protocol*. RFC 3530, IETF, April 2003. http://rfc.net/rfc3530.html.

[ODBC]
ODBC--Open Database Connectivity Overview, Micrsoft Knowledge Base Article 110093, http://support.microsoft.com/kb/110093.

[OGSA]
Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., Von Reich, J., *The Open Grid Services Architecture, Version 1.5*. Open Grid Forum OGSA-WG. GFD-I.080, July 2006. http://www.ggf.org/documents/GWD-I-E/GFD-I.80.pdf.

[OGSA BSP-Core]
Mori, T., and Siebenlist, F., *OGSA Basic Security Profile 1.0— Core*. Global Grid Forum OGSA-WG, Draft 5, January 2006. https://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-basic-security-profile-secure-channel/en/6

[OGSA BSP-Secure]
Mori, T., and Siebenlist, F., *OGSA Basic Security Profile 1.0— Secure Channel*. Global Grid Forum OGSA-WG, Draft 17, January 2006. https://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-basic-security-profile-secure-channel/en/17

[OGSA ByteIO]
Morgan, M *ByteIO Specification 1.0*. OGF GFD.87, January 2007.

[OGSA DMI]
Antonioletti, M., Drescher, M.,Luniewski, A. and Newhouse, S., *OGSA-DMI Functional Specification v1.0*. OGF OGSA-DMI-WG., in development. http://forge.gridforum.org/sf/go/doc14200?nav=1

[OGSA Glossary]
Treadwell, J. (ed.) *Open Grid Services Architecture Glossary of Terms, Version 1.5*. Open Grid Forum OGSA-WG. GFD-I.081, July 2006. http://www.ogf.org/documents/GWD-I-E/GFD-I.81.pdf.

[OGSA Profile Definition]
Maguire, T., Snelling, D. *OGSA Profile Definition*. Global Grid Forum, Lemont, Illinois, U.S.A., GFD-I.059, January 2006. http://www.ogf.org/documents/GFD.59.pdf

[OGSA Roadmap]
Kishimoto, H., and Treadwell, J. (eds.) *Defining the Grid: A Roadmap for OGSA Standards*. Global Grid Forum, Lemont, Illinois, USA, GFD-I.053, September 2005. http://www.ogf.org/documents/GFD.53.pdf.

[OGSA WSRF]
Foster, I., Maguire, T. and Snelling, D.: *OGSA WSRF Basic Profile 1.0*. GGF OGSA Working Group (OGSA-WG), GFD-R-P.072, May 2006. http://www.ogf.org/documents/GFD.72.pdf.

[PNG]
T. Boutell, et. al. *PNG (Portable Network Graphics) Specification*. Version 1.0. IETF. http://www.ietf.org/rfc/rfc2083.txt.

[POSIX]
IEEE Std. 1003.1-1990 *Standard for Information Technology Portable Operating System Interface (POSIX) - Part 1: System Application Programming Interface (API)*.

[RNS]
Pererira, M., Tatebe, O., Luan, L. and Anderson, T.., *Resource Namespace Service Specification*, GFDF-R-P-101, Open Grid Forum, March, 2007, http://www.ogf.org/documents/GFD.101.pdf.

[Scenarios]
Berry, D., Luniewski, A., and Davey, S., *OGSA Data Architecture Scenarios* GFDF-R-P-101, Open Grid Forum, March, 2007. . https://forge.gridforum.org/sf/go/doc14073?nav=1.

[SRM]
Sim, A. and Shoshani, A. *The Storage Resource Manager Interface Specification, version 2.2*, CERN, FNAL, JLAB, LBNL and RAL, April 2007, http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.pdf.

[SwA]
Barton, J., Thatte, S. and Nielsen, H. *SOAP Messages with Attachments*, W3C Note, December 2000, http://www.w3.org/TR/SOAP-attachments.

[TIFF]
TIFF Revision 6.0, Adobe, June 1992, http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

[UDDI]
Universal Description Discovery and Integration, OASIS published standard, version 2 of July, 2002, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

[URI]
Berners-Lee, T., Fielding, R., Irvine, U.C., Masinter, L. *Uniform Resource Identifiers (URI): Generic Syntax*. August 1998. http://www.ietf.org/rfc/rfc2396.txt.

[UTF-8]
Yergeau, F., *UTF-8, a transformation format of ISO 10646*. IETF. http://www.ietf.org/rfc/rfc3629.txt.

[WS-Addressing-core]
Gudging, M. Hadley, M. and Rogers, T. *Web Services Addressing 1.0 – Core*. W3C Recommendation May 2006. http://www.w3.org/TR/ws-addr-core.

[WS-Agreement]
Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S. and Xu, M. *Web Services Agreement Specification (WS-Agreement)*. Open Grid Forum GFD.107. May 2005. http://www.ogf.org/documents/GFD.107.pdf

[WS-AtomicTransaction]
Cabrera, L.F., Copeland, G., Feingold, M.,Freund, R.W., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I.,Storey, T., and Thatte, S. *Web Services Atomic Transaction, Version 1.0*, August 2005. http://xml.coverpages.org/WS-AtomicTransaction200508.pdf.

[WS-BusinessActivity]
Cabrera, L.F., Copeland, G., Feingold, M.,Freund, R.W., Freund, T.,Joyce, S., Klein, J., Langworthy, D., Little, M., Leymann, F., Newcomer, E., Orchard, D., Robinson, I., Storey, T., Thatte, S., *Web Services Business Activity Framework (WS-BusinessActivity),* OASIS Submission, August 2005. http://specs.xmlsoap.org/ws/2004/10/wsba/wsba.pdf

[WS-Context]
D. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, K. Swenson. *Web Services Context (WS-Context)* , http://www.oasis-open.org/committees/download.php/4344/WSCTX.pdf

[WS-Coordination]
F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Schwchuk and T. Storey, *Web Services Coordination (WS-Coordination)*, http://www-106.ibm.com/developerworks/library/ws-coor/.

[WS-Coordination Framework]
M. Little, E. Newcomer, G. Pavik. *Web Services Coordination Framework Specification (WS-CF)*. OASIS. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf.

[WS-DAI]
Antonioletti, M., Atkinson, M., Krause, A., Laws, S., Malaika, S., Paton, N. W., Pearson, D. and Riccardi, G., *Web Services Data Access and Integration – The Core (WS-DAI) Specification, Version 1.0*. GFD-R-P.074, Global Grid Forum. July 2006. http://www.ggf.org/documents/GFD.74.pdf.

[WS-DAIR]
Antonioletti, M., Collins, B., Krause, A., Laws, S., Malaika, S., Magowan, J., Paton, N. W., *Web Services Data Access and Integration – The Relational Realisation (WS-DAIR) Specification, Version 1.0*. GFD-R-P.076, Global Grid Forum. July 2006. http://www.ggf.org/documents/GFD.76.pdf.

[WS-DAIX]
Antonioletti, M. , Krause, A., Hastings, S., Langella, S., Laws, S., Malaika, S., Paton, N. W., *Web Services Data Access and Integration – The XML Realisation (WS-DAIX) Specification, Version 1.0*. GFD-R-P.075, Global Grid Forum. August 2006. http://www.ggf.org/documents/GFD.75.pdf.

[WS-DM]
Bullard, V., Murray, B. and Wilson, K. *An Introduction to WSDM*, OASIS wsdm-1.0-intro-primer-cd-01, February 2004. http://www.oasis-open.org/committees/download.php/16998/wsdm-1.0-intro-primer-cd-01.doc

[WS-Management]
Web Services for management (WS-Management), DMTF DSP0226, April 2006. http://www.dmtf.org/standards/wbem/wsman.

[WS-Naming]
Grimshaw, A. and Snelling, D. *WS-Naming Specification*. GFD 109, Open Grid Forum, July 2007. http://www.ogf.org/documents/GFD.109.pdf .

[WS-Policy]
Schlimmer, J., Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin,A., Nagaratnam, N., Prafullchandra, H., von Riegen, C., Roth, D., Sharp, C., Shewchuk, J., Vedamuthu, A., Yalçinalp, Ü., and Orchard, D., Web Services Policy 1.2 - Framework (WS-Policy), W3C submission, April 2006. http://www.w3.org/Submission/WS-Policy/

[WS-TransactionManagement]
Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K., *Web Services Transaction Management (WS-TXM) Ver1.0*, http://www.arjuna.com/library/specs/ws_caf_1-0/WS-TXM.pdf

[XIO]
Globus XIO. *The Globus eXtensible Input Output library*. http://www-unix.globus.org/api/c-globus-3.2/globus_xio/html/.

[XML]
Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and Yergeau, F. *Extensible Markup Language (XML) 1.0 (Third Edition),* W3C Recommendation 4th February 2004. http://www.w3.org/TR/REC-xml/.

[XSD]
Thompson, H., Beech, D., Maloney, M. and Mendelsohn, N. *XML Schema Part 1: Structures Second Edition,* W3C Recommendation October 2004. http://www.w3.org/TR/xmlschema-1/.