Editors:
I. Foster, ANL
S. Tuecke, ANL
J. Unger, IBM

August 14, 2003

OGSA Data Services

Abstract

This document describes a general framework for including data resources into the service-oriented Open Grid Services Architecture (OGSA). An OGSA data service is a Grid service that implements one or more of four base data interfaces to enable access to, and management of, data resources in a distributed environment. Data services are built on OGSI, which extends Web services to incorporate mechanisms for naming and reference of service instances, state management, notification, dynamic service creation, and lifecycle management. The base data interfaces, DataDescription, DataAccess, DataFactory, and DataManagement, define basic service data and/or operations for representing, accessing, creating, and managing data services. Data services implement various combinations of these interfaces, typically in extended forms, to incorporate information resources such as file systems and files, relational databases and tables, XML collections and documents, large binary objects (such as images or multi-media streams), and application generated data into the OGSA/OGSI service-oriented architecture.

Table of Contents

1   Introduction

A service-oriented treatment of data can allow data to be treated in the same way as other resources within the Web/Grid services architecture. Thus, for example, we can integrate data into registries and coordinate operations on data using service orchestration mechanisms. A service-oriented treatment of data also allows us to exploit Open Grid Services Architecture (OGSA) mechanisms [3] when manipulating data. For example, we can use Open Grid Services Infrastructure (OGSI) Grid Service Handles as global names for data, manage the lifetime of dynamically created data by using OGSI lifetime management mechanisms, and represent agreements concerning data access via OGSI-Agreement.

The design of appropriate interfaces and behaviors for such "data services" is made complicated by the heterogeneous nature of the data sources and data access methods found in distributed systems. In an environment that features data maintained in or produced by file systems, databases, object stores, sensors, etc., it is not sufficient simply to specify a "data service" interface that defines, via standard "getData" and "putData" operations, a single view of different data sources. For example, depending on context, we may want to interact with the contents of a particular file system as a directory, relational database, row in a relational table, or sequence of bytes.

Recognizing this need to embrace and expose diversity, we present a service-oriented treatment of data that allows for the definition, application, and management of diverse abstractions—what we term *data virtualizations*—of underlying data sources. ("Data virtualization" is one of a number of terms for which we adopt specific meanings within this document. See Table 1 for definitions, and references to more detailed discussions.) This material has been prepared as a contribution to the work of the Global Grid Forum's OGSA Data Access and Integration Services (DAIS) work group [4].

In our service-oriented treatment of data, a data virtualization is represented by, and encapsulated in, a *data service*, an OGSI Grid service with *service data elements* (SDEs) that describe key parameters of the virtualization, and with *operations* that allow clients to inspect those SDEs, access the data using appropriate operations, derive new data virtualizations from old, and/or manage the data virtualization. For example, a file containing geographical data might be made accessible as an image via a data service that implements a "JPEG Image" virtualization, with SDEs defining size, resolution, and color characteristics, and operations provided for reading and modifying regions of the image. Another virtualization of the same data could present it as a relational database of coordinate-based information, with various specifics of the schema (e.g., table names, column names, types) as SDEs, and SQL as its operations for querying and updating the geographical data. In both cases, the data service implementation is responsible for managing the mapping to the underlying data source.

Having embraced diversity, it becomes important to identify and provide common representations for common core behaviors and to define clearly what is (and what is not) a "data service." To this end, we (a) define four *base data interfaces* (WSDL portTypes) that can be used to implement a variety of different data service behaviors, and (b) specify that a data service is any OGSI-compliant Web service that implements one or more of these base data interfaces.

The four base data interfaces are as follows. We show below how these base interfaces can be combined and extended to define various interesting services.

- *DataDescription* defines OGSI service data elements representing key parameters of the data virtualization encapsulated by the data service.

- *DataAccess* provides operations to access and/or modify the contents of the data virtualization encapsulated by the data service.

- *DataFactory* provides an operation to create a new data service with a data virtualization derived from the data virtualization of the parent (factory) data service.

- *DataManagement* provides operations to monitor and manage the data service's data virtualization, including (depending on the implementation) the data sources (such as database management systems) that underlie the data service.

As we describe below, our definitions for these services build on and extend not only core OGSI interfaces (GridService and Factory) but also OGSI-Agreement interfaces [1], which are used to incorporate agreements (e.g. Quality of Service guarantees, payment information, etc.) into the various data operations. We also expect that (yet-to-be-defined) OGSA relationship management services will be used to represent and manage relationships among virtualizations, such as multiple virtualizations against the same data source, and dependencies between virtualizations.

Figure 1 summarizes the architecture and overall scope of the OGSA data service concept. In the rest of this document, we first discuss data virtualizations in more detail (Section 2), then describe the four base data interfaces (Section 3), and then discuss various other aspects of the data service concept.

Table 1: Key terms used when describing OGSA data services, and their definitions.

Term	Definition	Examples	See
Data virtualization	An abstract view of some data, as defined by operations plus attributes (which define the data's structure in terms of the abstraction) implemented by a data service.	A (virtual) file system, JPEG file, relational database, column of a relational table, random number generator.	§2.1
Base data interface	DataDescription, DataAccess, DataFactory, and DataManagement define mechanisms for inspecting, accessing, creating, and managing data virtualizations, respectively. They are expected to be extended to provide virtualization-specific interfaces.	Extensions of the base data interfaces might include RelationalDescription, SQLAccess, FileFactory, and FileSystemManagement.	§3
Data service	An OGSI-compliant Web service that implements one or more of the four base data interfaces, either directly, or via an interface that extends one or more base data interfaces, and thus provides functionality for inspecting and manipulating a data virtualization.		§3
Data set	An encoding of data in a syntax suitable for externalization outside of a data service, for example for communication to/from a data service.	WebRowSet XML encoding of SQL query result set, JPEG encoded byte array, ZIP encoded byte array of a set of files.	§3.2

Data source	A necessarily vague term that denotes the component(s) with which a data service's implementation interacts to implement operations on a data virtualization.	A file, file system, directory, catalog, relational database, relational table, XML document, sensor, or program.	§2.1
Resource manager	The logic that brokers requests to underlying data source(s), via a data virtualization, through the data interfaces of a data service.	An extension to, or wrapper around, a relational DBMS or file system; a specialized data service.	§2.3
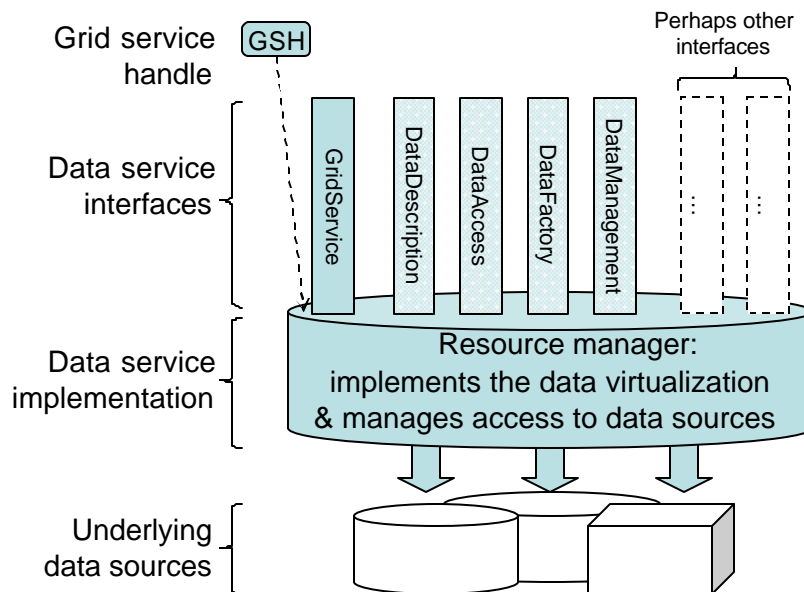
Figure 1: Architecture and scope of the OGSA data service concept. The shaded areas denote a data service, the GridService and four base data interfaces, and a Grid Service Handle that references the data service. The service's implementation (sometimes referred to as a "resource manager") brokers requests to underlying data source(s), via the service's data virtualization, through the data interfaces.

2   Data Virtualizations

The data virtualization abstraction is fundamental to our approach to OGSA data services, and so we provide a more detailed discussion of the concept.

2.1 The Need for Virtualization

A distributed system may contain data maintained in different syntaxes, stored on different physical media, managed by different software systems, and made available via different protocols and interfaces. We use the general term *data source* to denote a system- or implementation-specific physical or logical construct that provides access to data. Examples of a data source include an individual file, a file system, a directory, a catalog, a relational database, a relational table, an XML document, and a large binary object (BLOB). A sensor that responds to a query by making a physical measurement, and a program that responds to a query by computing

a value, can also be viewed as data sources. A data service can itself be a data source for another data service.

While different physical media and storage management systems have their own peculiarities, service-oriented interfaces can be defined and implemented that make any particular data source accessible to clients in a wide variety of ways. For example, given a JPEG image stored in a file or relational database, we might define service interfaces that make it accessible as:

- one file in a larger file system virtualization (with associated operations for manipulating files in the file system);

- one file in a larger file set comprising multiple JPEG images that together form a movie (with associated operations for playing the movie);

- a JPEG image of a particular size, resolution, and color characteristics (with associated operations for reading or modifying regions of the image),

- a set of relational tables representing the features and components of the image (with SQL operations for accessing those tables), and/or

- a sequential array of bytes (with associated Posix-style operations for reading and writing the file).

Each abstraction of the underlying data has different performance characteristics, depending for example on how closely the abstraction corresponds to the underlying storage system's representation of the data (e.g., is it a file or database?). Regardless of performance considerations, different abstractions can be useful in different situations.

We introduce the term *data virtualization* to denote a particular service-oriented interface to data from one or more data sources. The abstraction that a data virtualization provides of its underlying data can be simple (e.g., a straightforward service-oriented rendering of the underlying storage system's interface) or complex (e.g., a transformation from files to tables); may correspond to a subset of an individual data source (e.g., a view on a database or file within a file system) or federate multiple data sources and/or services; and can involve simple data access or computational transformations of underlying data.

Mappings between data virtualizations and underlying data sources and services may be one-to-one, many-to-one, one-to-many, or many-to-many. A many-to-one mapping can occur when a data source is virtualized simultaneously at different levels of granularity (see Figure 2). For example, a file system might support data virtualizations for the file system as a whole (with associated operations for managing the file names and metadata); arbitrary subsets of files in the file system (with associated operations for modifying or accessing all files in the set as a whole), and/or individual files (with associated operations for reading and writing the contents of the file). A many-to-one mapping can also occur when different service interfaces are defined to the same underlying data virtualization that provide different subsets of available functionality—perhaps for reasons of access control.

In the case of a many-to-one or many-to-many relationship, multiple data virtualizations may refer to the same underlying data sources. Thus, an update to one data virtualization may also result in updates to others. For example, in Figure 2, the Movie refers to the same underlying physical storage as the various Frames. Modifying a Frame also modifies the Movie. OGSA relationship services (yet-to-be-defined) may be used to represent such relationships so that clients can discover that, for example, a particular Frame is part of a particular Movie.
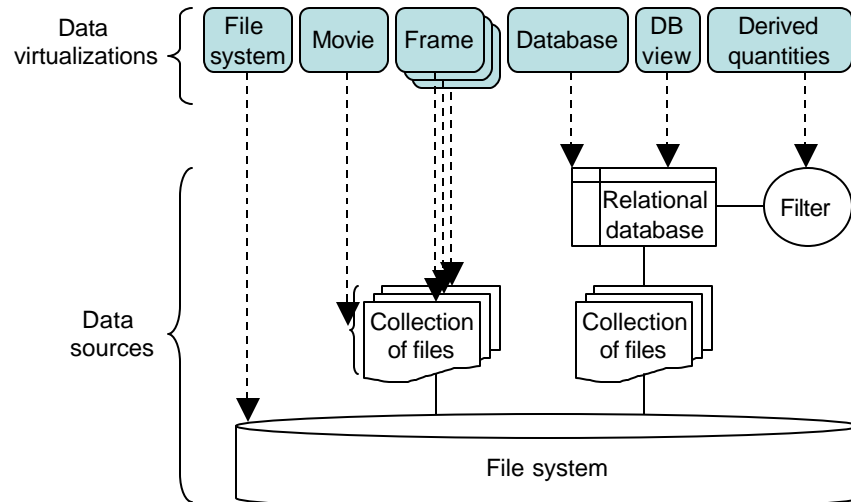
Figure 2: An illustration of how different data virtualizations can provide different views of the same or different parts of a data source.

2.2 Representing Data Virtualizations

As noted above, a data virtualization is represented by a *data service*, an OGSI-compliant Web service that implements one or more of the base data interfaces.

The term *OGSI-compliance* means simply that the service is a Web service that (a) implements the OGSI GridService portType, which provides lifetime management and "service data elements" (SDEs) for service inspection and monitoring, and (b) has a Grid Service Handle that uniquely names that service [5]. Thus, any data service has a globally unique name and SDEs that allow for the discovery of attributes (both metadata and state) of the service. A particular data service may of course also implement other OGSI interfaces, such as OGSI service data notification subscription operations).

We exploit OGSI mechanisms within our OGSA data service framework in a variety of ways, as we now describe.

2.2.1 Service Data to Represent Data Service State

We use the OGSI SDE mechanism to describe aspects of a data service's data virtualization, such as table names, column names, types, and number of rows in a relational data virtualization, or file names and sizes in a file system data virtualization. SDEs may also be used to describe "metadata" about the data virtualization, such as who produced the data, its purpose, and abstract identifiers and properties of portions of the data. This use of SDEs enables inspection and discovery via standard mechanisms. We will probably also want to standardize the SDEs used within various specific domains. Depending on context, this standardization could occur within GGF, DMTF, discipline-specific standard bodies, etc.

2.2.2 Grid Service Handles as Global Names

OGSI compliance also means each data service (and thus its virtualization) has a name (the URI-encoded OGSI Grid Service Handle) that is globally unique for all time. This unique identity can be used in many different ways: for example, as a handle to be shared amongst cooperating clients; as a basis for associating externally managed metadata with the virtualization; and (when

presented to a handle resolution service) as a means of obtaining the information (a Grid Service Reference: GSR) needed to communicate with the data service.

The power and generality of this global naming scheme makes it good practice to use GSHs whenever global names are needed within an OGSA application. For example, consider a listFiles operation on a "directory" data service. As we discuss in more detail in Section 3.3.1, if this operation is intended to return names with other than local scope, then an appropriate OGSA implementation will return a set of GSHs naming the files contained within that directory. These GSHs can then be passed to other clients, queried for metadata, and used to access the underlying files.

Despite the many advantages of GSHs as global names, concerns about the cost of Grid services and GSHs can lead developers to consider auxiliary global naming schemes for particular purposes. Thus, for example, a developer might define a listFiles operation that returns not GSHs but strings denoting file names, and then pass around <GSH, file -name> pairs as names for files. In effect, the developer creates an auxiliary global naming scheme. However, the resulting lack of uniformity provides no significant advantages (assuming that OGSA implementations uses appropriate techniques to represent data services efficiently) and can lead to significant difficulties and complexities in both client and service implementations.

2.2.3 Lifetime Management of Data Services and Sources

OGSI lifetime management mechanisms can be used to manage the lifetime of data services and also perhaps, depending on context, their underlying data source(s).

While some data services will be created via out-of-band mechanisms and will map to persistent data sources (see Section 4), others can be created dynamic ally by data factories, and/or may have finite lifetimes managed via OGSI lifetime management mechanisms. That is, an initial lifetime may be established at the time of creating the data service, this lifetime may be modified, and/or the service may be destroyed via OGSI operations. The *meaning* of this service lifetime will depend on the service definition (i.e., the semantics associated with its interfaces) and its implementation. In some situations, it may be simply the *service* that is created and destroyed, while the underlying data source(s) persists; in others, the lifetime of the data service and the underlying data source(s) may be tightly coupled, with (for example) a database management system being started or a file being created when the data service is created, and that database management system or file being destroyed when the data service itself terminates.

2.2.4 Representing Sessions as Transient Services

Data services exploit transient Grid services (i.e., services that are created by an OGSI Factory, and that have limited lifetimes) to manage "sessions" with particular clients. For example, a client of a relational database may want to perform a select operation against the database, and then incrementally retrieve portions of the result set through a cursor-based access interface. Assuming the relational database is virtualized as a data service, the DataFactory interface of this service could be used by the client to create a new data service whose virtualization is the result set. The client can then interact with this new data service through a series of operations to incrementally retrieve data sets that represent portions of the complete result set. The new data service both represents and manages the cursor-based access session with the client.

A data service may, like any Grid service, be accessed by many clients and, like any Grid service, may deal with concurrent requests in a variety of ways, including sequentializing the processing of requests, allowing for concurrency, and/or providing for concurrency control mechanisms in its interface.

2.3 Implementation

The mapping from a data virtualization to its underlying data source(s) is determined by the data service's implementation. A data source that is "virtualized" by a data service is encapsulated by the data service's implementation and is not visible or accessible to users of that data other than by that service's operations. Data service *management* interfaces can be an exception to this encapsulation: these interfaces may need to reference the identity and schema of a data source to create and manipulate data virtualizations of a data source.

Not withstanding the statements made in the first paragraph, a data source may of course allow access to its data via non-OGSA mechanisms, including data system-specific interfaces such as file I/O and JDBC. However, this access occurs outside this OGSA data services model.

The concept of a resource manager can be useful when describing a data service implementation. A resource manager mediates access to the data sources encapsulated by the data service to provide a "virtualized" instance of that data as part of the Web/Grid Service infrastructure. A particular resource manager may be an extension to an existing data system such as a relational or XML DBMS, file system, document store, content management system, or other specialized data source. Alternatively, it may be a separate layer implemented on top of such a system to present a "service" interface to the system. The resource manager is not an architecturally prescribed part of the data service model, but it can nonetheless be useful in describing data services and how they relate to existing data systems.
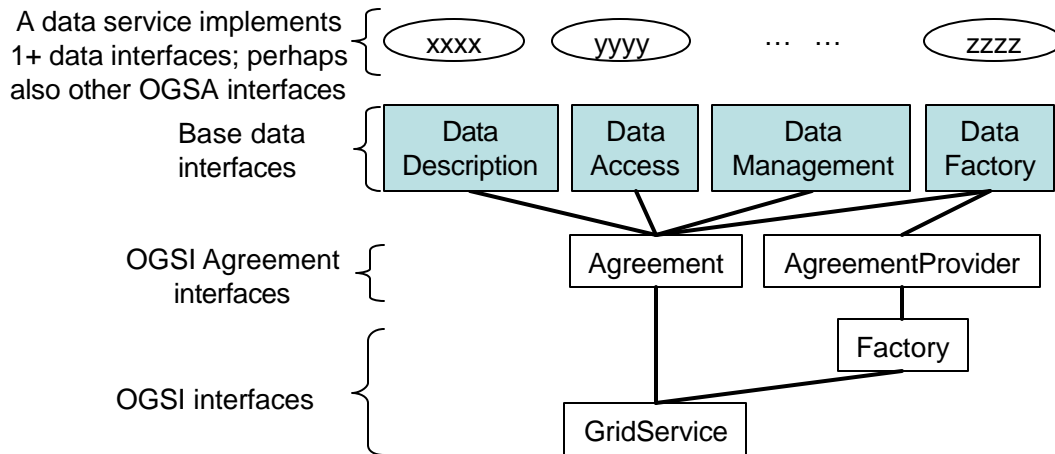
3 Data Services and Data Interfaces

A data service implements interface(s) and associated behavior(s) for the manipulation of data virtualizations. More specifically, a data service is an OGSI-compliant Web service that implements (either directly, or via some extended version) one or more of the four base data interfaces.

Each of the four *base data interfaces*—DataDescription, DataAccess, DataFactory, and DataManagement—defines service data elements and/or operations that can be invoked against a data virtualization. As illustrated in Figure 3, each base data interface extends an OGSI-Agreement interface, which in turn extends OGSI GridService. And, as shown in Figure 4, each data interface may in turn be specialized for particular types of data virtualizations. For example, extended versions of DataDescription might include RelationalDescription, WebRowSetDescription, FileSystemDescription, and FileDescription.

A data service may implement various combinations of these data interfaces. For example, a simple data service that virtualizes a file might implement just a FileDescription interface that describes the file, along with a FileAccess interface for reading and writing sections of the file. A more complex data service might support interfaces that describe and allow access to its virtualization in multiple ways. For example, if the virtualized file is a JPEG image, then in addition to the FileDescription and FileAccess interfaces, the data service might also implement JPEGDescription and JPEGAccess interfaces to allow more specialized description and access.

In the rest of this section, we describe the four base data interfaces in more detail, including examples of extensions to each of the base data interfaces that are specialized for particular purposes. See also Table 2, which summarizes key features of these interfaces.

Figure 3: Data services (at top) implement one or more of the base Data interfaces (shaded) or extended versions of those interfaces. The Data interfaces themselves extend OGSI and OGSI-Agreement interfaces. (Lines linking interface names represent interface extension.)
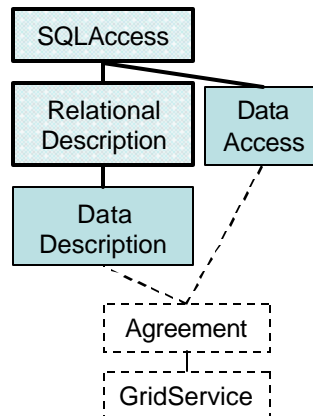
Figure 4: An example of how base data interfaces (in this case DataDescription and DataAccess) can be extended to define interfaces specific to a particular data virtualization.

Table 2: Synopsis of the base data interfaces.

Interface	Extends	Service Data (in addition to that in extended interface)	Operations (in addition to those in extended interface)
Data Description	Agreement	Parameters describing the data service's data virtualization.	None.
Data Access	Agreement	Information about the current state of the requested access or update.	Access data contained within the data virtualization. Update data contained within the data virtualization.

Data Factory	Agreement Provider	Descriptions of the parameters that may be passed to the Factory CreateService operation for configuring the derived data virtualization	None. (The CreateService operation is inherited from Factory, via AgreementProvider.)
Data Management	Agreement	Information about the data virtualization that can monitored, such as performance.	Configuration of the data service.

3.1 DataDescription

The DataDescription interface defines SDEs that describe the data virtualization supported by a particular data service. These SDEs are typically used to inform clients about the details of the service's data virtualization, so that the client can formulate appropriate requests to DataAccess, DataFactory, and/or DataManagement interfaces supported by that service. These SDEs may be queried through OGSI-defined service data operations such as FindServiceData and notification subscription. An extended DataDescription interface might also define operations beyond the OGSI service data operations in order to provide more specialized inspection capabilities.

The base DataDescription interface will presumably define SDEs that are assumed to be common across all data virtualizations. These SDEs must be defined in future work. More specialized interfaces may extend the base DataDescription interface to introduce SDEs that are relevant to specific data virtualizations, for example defining a data virtualization's schema along with generic properties and associated policies. Extended description interfaces may be generic (e.g., a RelationalDescription interface that defines SDEs describing database names, table names, column names, and column types) or domain-specific (e.g., a climate modeling or financial analysis data virtualization).

The DataDescription interface extends the Agreement interface from OGSI-Agreement, which in turn extends the OGSI GridService interface. The GridService interface provides (among other things) SDEs which allow for the discovery of the interfaces supported by the Data Service, such as the various extensions of the DataDescription, DataAccess, DataFactory, and DataManagement interfaces. The Agreement interface allows for inspection, monitoring, and possibly re-negotiation of the agreement terms governing use of the data service.

We expect the GGF DAIS working group and others to develop specifications that define the DataDescription interface and various extensions of it. The following examples are intended only to be illustrative of the extensions that might be defined in such specifications.

RelationalDescription defines SDEs for describing the schema of a relational database, including table names, column names and types, table sizes, and associated attributes (e.g., ownership, and permissions).

RowSetDescription defines SDEs that describe a set of rows (e.g., column names, column types, number of rows), such as may result from an SQL select.

XMLCollectionDescription defines SDEs that describe a collection of XML documents, including their XML Schemas and number of documents.

FileSystemDescription defines SDEs that describe a file system, including directories and filenames, and attributes of each (e.g., ownership, permissions, and modification time).

FileDescription defines SDEs that describe a file, including the size of the file.

3.2 DataAccess

The DataAccess interface provides operations to access and modify the contents of a data service's data virtualization. This interface extends the Agreement interface from OGSI-Agreement, which in turn extends the OGSI GridService interface. The GridService interface provides (among other things) operations for managing the lifetime of the data service. The Agreement interface allows for inspection, monitoring, and possibly re-negotiation of the agreement terms governing use of this data service.

Extensions of the DataAccess interface contain operations that are specialized for access to particular types of data virtualizations. The base DataAccess interface will presumably define SDEs and operations that are assumed to be common across all data virtualizations. These SDEs and operations must be defined in future work.

In describing the DataAccess interface, we use the term *data set* to denote an encoding of data in some machine-readable form such as XML. Note that in this terminology, a data set is a *syntax*, not a service. Data sets typically appear as input and output parameters to the DataAccess interfaces, often as an externalization of part of a data virtualization. For example, an SQLAccess interface might have an Select operation that takes an SQL select statement as input, and produces as output a data set containing the resulting rows, encoded as an XML element conforming to the WebRowSet XML Schema.

We expect the GGF DAIS working group and others to develop specifications that define the DataAccess interface and various extensions of it. The following examples are intended only to be illustrative of the extensions that might be defined in such specifications.

> *SQLAccess*: SQL based queries and updates to relational data virtualizations. This interface may be further extended to support various extensions to SQL, such as those provided by specific database products. (Alternatively, the SQLAccess interface could be made extensible to support and describe the various SQL extensions directly.) These interfaces will often be used in conjunction with the RelationalDescription interface. The operations supported by this interface would be "stateless," meaning, for example, that queries would return the entire result as a data set (i.e., an encoded representation of the result set), and updates would take as input a data set with the data to be updated in the database. To create a result set that can be retrieved incrementally, for example via a "stateful" cursor-based access interface, the SQLFactory interface would instead be used to create a new data service which contains the result set (perhaps virtually) and which implements the RowSetDescription and CursorRowSetAccess interfaces.

> *CursorRowSetAccess*: Cursor based access to a row set data virtualization. This interface will often be used in conjunction with the RowSetDescription interface. This is a "stateful" interface, meaning that one operation may affect the behavior of future operations: an operation to get the next N rows of the row set will update the cursor in the row set so that subsequent operations (by the same client or another) will not get the same rows.

> *XMLCollectionAccess*: XPath-, XQuery-, and XUpdate-based access to an XML Collection data virtualization. This interface will often be used in conjunction with the XMLCollectionDescription interface. Like SQLAccess, this is a "stateless" interface, and there is an XMLCollectionFactory companion interface for creating "stateful" data services for retrieving results incrementally.

> *StreamAccess*: Incremental read and write operations against a byte stream data virtualization. This interface will often be used in conjunction with the FileDescription

interface. Like CursorRowSetAccess, this is a "stateful" interface that will typically reside on a data service that is created by a DataFactory (e.g., FileSelectionFactory).

FileAccess: This extension to StreamAccess allows for Posix-style, file-pointer based incremental read and write to a file data virtualization, including the ability to reposition the pointer. This interface will often be used in conjunction with the FileDescription interface. Like CursorRowSetAccess, this is a "stateful" interface that will typically reside on a data service that is created by a DataFactory (e.g., FileSelectionFactory).

BlockAccess: Block (file position and size) read and write operations against a file data virtualization. This interface will often be used in conjunction with the FileDescription and FileSystemDescription interfaces. This is a "stateless" interface.

TransferSourceAccess, *TransferSinkAccess*: Endpoints for multi-protocol, third-party data transfer between two data services. The TransferSourceAccess interface is a "stateful" interface for configuring and managing the generation of a series of data sets that together comprise the entire data virtualization. These data sets are delivered to another data service that implements the TransferSinkAccess interface. This is an OGSI-compliant generalization of the capabilities found in GridFTP.

We expect other, more domain-specific, extensions to DataAccess to be defined, but probably not by the GGF DAIS working group. For example, a 2DImageAccess interface would allow for images (e.g., JPEG encoded images) to be accessed in whole or part (e.g., a region of the image), perhaps with various transforms applied (e.g., change resolution, make it black and white, etc). A corresponding 2DImageDescription interface would allow for the description of the image, including resolution, color depth, registration information, etc.

A single data service may simultaneously implement multiple extended DataAccess interfaces, thus allowing for multiple access approaches to the service's data virtualization.

3.3 DataFactory

The DataFactory interface supports a request to create a new data service whose data virtualization is derived from the data virtualization of the parent data service (the one that implements the DataFactory). The "derivation" used to generate the new data virtualization can range from a simple subsetting to a complex transformation. The interfaces supported by the new data service may be the same as, or different from, those of the parent (factory) data service.

Extensions of the base DataFactory interface can define Factory parameterizations specialized for derivations from and to particular data virtualizations. The base DataFactory interface will presumably define SDEs and operations that are assumed to be common across all data virtualizations. These SDEs and operations must be defined in future work.

The DataFactory interface extends the OGSI-Agreement AgreementProvider interface, which in turn extends OGSI Factory. A request to a DataFactory (like any OGSI Factory) results in the creation of a new data service, and the return of the Grid Service Handle (GSH) of that new service to the requestor. That GSH can then be used by requestors to direct requests at any interface implemented by the new service, including the various data service interfaces. As with any OGSI-compliant service instance, new data service instances created by a DataFactory have their own lifetime, service data, and state, and may be relatively transient or long-lived.

3.3.1 Use of DataFactory

The DataFactory interface is typically used for one of the following three reasons.

To create a name for a derived data virtualization. Each data virtualization, by virtue of being represented as a Grid service, has a name (its GSH) that is globally unique for all time. Access interfaces may be used to produce and consume data sets related to, or derived, from this virtualization, using parameters that are meaningful only within the context of operations against that particular virtualization. But a client may want to obtain a more portable name for a particular derivation of a data virtualization, for example, to give to another client. In this situation, a DataFactory should be used to create a new "derived data virtualization" as a separate data service, so that the derived data virtualization has a name (its GSH), and can be managed using standard OGSI mechanisms. Examples of such derived data virtualizations include creating a view of a relational database, creating a result set from a database query, and creating a file set from a file system. Note that use of a DataFactory does *not* imply anything about the materialization of the derived data virtualization. Rather, the new data service may just be a virtual façade over the same underlying data source as the parent data service, and data may be materialized lazily as necessitated by operations against the derived data service.

To create a session for a client. Some forms of data access require "stateful" interactions between a client and the data source. For example, when posing a query against a database that is expected to create a large result set, a client may wish to establish a session within which it can incrementally retrieve portions of the result set. In this situation, a DataFactory should be used to create a new data virtualization representing the data for the session (e.g., the result set), with a set of DataAccess interfaces that are appropriate for that session (e.g., incremental retrieval of portions of the result set). Note that such sessions are not merely a convenience for the client, but may also be used by the data service implementation to optimize the data access operations. For example, in the above example the data service representing the result set could lazily produce the portions of the result set on-demand, so as to avoid materializing the complete result set at once.

To create an "empty" data virtualization. Suppose that a client wants to add a new file to a file system. This action could be done using a "stateless" DataAccess interface against the file system data service—for example, via an operation that has input parameters for the name of the file and the contents of the file. However, there are many times when it is useful to create an "empty vessel" in the form of a data service, into which data can be placed. For example, the file system data service could have a DataFactory that adds a new file to the file system, and creates a data virtualization for that new, empty file. The new data service for that file can then be used by clients to populate the file with its data, perhaps through a "stateful" DataAccess interface that accepts data incrementally.

3.3.2 DataFactory's Use of AgreementProvider

As an extension of AgreementProvider, all requests to a DataFactory for creation of a data service are defined using the Agreement language defined by OGSI-Agreement. This language is a framework for defining and negotiating all of the "terms and conditions" relevant to the creation and operation of a service. OGSI-Agreement does not define the exact terms of the Agreement, but rather expects domain-specific interfaces that extend AgreementProvider to define domain specific terms that populate an Agreement document. Thus, as an extension of AgreementProvider, the DataFactory interface defines Agreement terms that are relevant to all data services, while each DataFactory extension defines additional terms that are relevant to the specific data services that are created by that DataFactory extension. We discuss below the implications of using OGSI-Agreement within data services.

3.3.3 Extending DataFactory

We expect the GGF DAIS working group and others to develop specifications that define the DataFactory interface and various extensions of it. The following examples are intended only to be illustrative of the extensions that might be defined in such specifications.

FileSelectionFactory: Suppose the parent data service contains a data virtualization of a file system, as described by the FileSystemDescription interface. The FileSelectionFactory interface on this data service would allow for the creation of other file system data services containing a subset of the files in the parent data service. The FileSelectionFactory interface would also allow for the creation of data services containing a data virtualization of a single file, with specialized DataAccess interfaces for access that file, such as the above described FileAccess, StreamAccess, TransferSourceAccess, and 2DImageAccess (e.g., if the file contains a JPEG image).

SQLFactory: Suppose the parent data Service contains a data virtualization of a relational database, as described by the RelationalDescription interface. The SQLFactory interface on this data service would allow for the creation of a new data service whose data virtualization is relational subset or view of the parent's relational database, and that supports the RelationalDescription interface and appropriate SQLAccess and SQLFactory interfaces. The SQLFactory interface would also allow for the creation of a new data service whose data virtualization is a result set from an SQL select command, and that supports the RowSetDescription and CursorRowSetAccess interface. The SQLFactory interface would similarly allow for the creation of data services that can be used for incremental, or "stateful" update to the relational database.

XMLCollectionFactory: This specialized DataFactory is similar to SQLFactory, but for XML Collections. One interesting form of derived data service that the XMLCollectionFactory might create is one where selected XML elements from the XML collection are made available through standard OGSI service data access operations in the child data service, perhaps in addition to more specialized XML DataAccess interfaces.

TransferFactory: This factory creates a data service that implements the TransferSourceAccess and/or TransferSinkAccess interface(s), for bulk movement of the data contained in the data virtualization of the parent data service.

CollectionSelectionFactory: Suppose the parent data service contains a data virtualization that is a collection of all data sources available on a particular machine, including relational databases, XML databases, file systems, and specialized data collection instruments such as electron microscopes or sensors. The CollectionSelectionFactory interface on this data service would allow for the creation of a new data service containing a data virtualization of one of the data sources, with the appropriate specialized DataDescription and DataAccess interfaces on the new data service.

We emphasize again that the creation of a data service via a DataFactory does *not* necessarily imply materialization of the new data virtualization. The new data service may just be another virtualization over the same underlying data source(s). For example, creating a file virtualization using the FileSelectionFactory, or creating a view on a relational database using the SQLFactory, does not require (or prohibit) any data creation, copying, or movement. Similarly, a derived data service may defer the execution of any transformation until its data is needed. For example, using the SQLFactory to query a relational database to create a new data service for the result set does not imply that the actual query has completed, or even begun. Rather, the implementation of the derived data service might (either of its own volition, or under the control of configuration parameters passed to the factory operation) defer execution of the query and materialization of the

row set until operations are invoked by clients against its CursorRowSetAccess interface requesting portions of the result set.

3.3.4 Federating Multiple Data Sources

The DataFactory only allows for the derivation of new data virtualizations from the parent's data virtualization. It does not directly support the ability to create a data service whose data virtualization is a derivation of data virtualizations contained in two or more distinct data services. If desired, the latter behavior must be synthesized from a combination of the DataAccess, DataFactory, and DataManagement interfaces.

For example, to create a relational data virtualization that federates data from several relational data services, a DataFactory might first be used to create a new federated data service with an empty relational data virtualization. Then a combination of DataAccess and DataManagement operations could be used to populate the new data virtualization. DataAccess operations against the underlying data services could be used to retrieve data sets that could in turn be fed into DataAccess operations against the new data service, resulting in one-time copies of data from underlying data services into the federated data service. Alternatively, or in addition, DataManagement operations against the new data service could be used to define mappings between portions of the federated data service's virtualization and those of the underlying data services, so that when subsequent DataAccess operations are invoked against the federated data service, that would result in the invocation of the appropriate DataAccess operations against the underlying data services.

These approaches to creating data virtualizations that derive from multiple services may sound complicated. However, the underlying mechanics can typically be hidden in a data service's implementation. For example, a particular virtual organization (VO) might maintain a "VO data service" whose virtualization is a collection of all other data services within that VO. A specialized DataFactory could allow for various derived data virtualizations from this VO data service, including virtualizations that span the underlying data virtualizations contained within this VO data service.

3.4 DataManagement

The DataManagement interface provides operations to manage the data virtualizations (and indirectly the data sources that underlie them) of a data service. Extensions of the DataManagement interface contain operations that are specialized for management of particular types of data virtualizations. The base DataManagement interface will presumably define SDEs and operations that are assumed to be common across all data virtualizations. These SDEs and operations must be defined in future work.

The central purpose of a DataManagement interface is to allow clients (i.e., managers) to specify exactly how a data virtualization is constructed from one or more underlying data sources. That is, it allows for the specification of the projections, transformations, and federations that comprise a data virtualization. For example, a DataManagement interface to a relational database system might allow for specific tables from databases in the underlying data source to be made part of a data virtualization, or for views on the underlying database to be made available as tables within a data virtualization.

The DataManagement interface further may provide operations for configuring the base access policies of a data virtualization, thus bounding the agreements that requestors may negotiate with the data services. The DataManagement interface may also define SDEs related to the data virtualization, data sources, and the resource manager, so that client can monitor the data service's status, performance, etc.

Various extended DataManagement interfaces may affect only the data virtualizations implemented by a service, or they may additionally affect the underlying data source. For example, a RelationalDatabaseManagement interface implemented by a relational DBMS may allow both the definition of data virtualizations from the underlying data source (for example, making a particular database available as a data virtualization within a data service), as well as the creation and modification of databases in the underlying data source (for example, add new tables to a database in the DBMS).

Management interfaces are out of scope of GGF DAIS working group, but are nonetheless a critical part of the overall data services architecture. We expect a variety of extended DataManagement interfaces to be defined, some standardized through working groups in GGF and other standards organizations, and some product-specific.

4   Root Data Services

The abstractions and interfaces that we have described allow for the creation, destruction, composition, and federation of data service instances that manipulate data virtualizations. However, just as some Grid services are created and destroyed via mechanisms other than OGSI Factory and GridService lifetime management mechanisms, so we also must allow for "root" data services that are not instantiated dynamically but instead define data service interfaces to data systems that are managed via other means. These system-specific data systems include, for example, file systems, document stores, and database management systems. They either implement data services natively as a new method for accessing their managed data resources, or have data service implementations "wrapped" around their core implementations. In either case, they have an existence and lifetime that is external to OGSA.

Such root data services are created by some out-of-band means and project the complete underlying data source into one or more data virtualizations, each wrapped by a data service. The handles (GSHs) of these root data services are either discoverable via, for example, virtual organization registries, or are configured into applications and clients as a root address by some other out-of-band means. Other dynamic data services can then be derived via the DataFactory interface from these "persistent" data services.

5   Use of OGSI-Agreement

The draft OGSI-Agreement specification [1] builds on concepts introduced within the Service Negotiation and Access Protocol [2] and WSLA [ref] to define a general framework for the expression and negotiation of an agreement that contains the "terms and conditions" governing a service's creation and operation, relative to specific consumers of that service. OGSI-Agreement consists of 3 basic components. (1) An agreement language provides an extensible framework for expressing the terms of an agreement between consumer and service provider, including what actions are to be performed, under what guarantees (i.e., qualities of service: QoS), how compliance will be determined, and payment terms. (2) The AgreementProvider interface extends the OGSI Factory interface to define how the Factory CreateService operation is used with the agreement language to instantiate an agreement with a service provider, where the GSH of the created service uniquely identifies the agreement. (3) The Agreement interface extends the OGSI GridService interface and must be implemented by the service created by an AgreementProvider. It provides operations for the monitoring and re-negotiation of the terms of the agreement.

OGSI-Agreement is a critical foundation to the data services described here. Access to all data services is governed by agreements, as defined by OGSI-Agreement. While all agreements include terms describing the action(s) to be performed (e.g., a query), other agreement terms may

range from simple best-effort performance of the requested actions, to strict guarantees about their performance with remuneration for failure to comply. The DataFactory interface extends the AgreementProvider interface, while DataDescription, DataAccess, and DataManagement extend the Agreement interface. Thus, all requests to create a data service via DataFactory are also requests to instantiate an agreement that governs access to the new data service. Agreements created by DataFactory may be complete, stand-alone agreements, or may reference (and perhaps extend) pre-arranged agreements.

The parameters to DataFactory are expressed as terms of the agreement language – that is, individual XML elements that extend the OGSI-Agreement base term type, and which can be combined in a single OGSI-Agreement defined agreement element. Terms may be general or specific to an extended DataFactory, and may address, for example, how the new data virtualization should be derived from the parent (factory) data virtualization; what base data interfaces (and other interfaces) are required or desired on the new data service; performance and quality characteristics (QoX) of new data service, including QoS quality of data retention (QoR), quality of data (QoS), and quality of protection (QoP); how will the agreement be monitored for compliance with these terms; and billing information.

6 Example Data Services

We use some simple examples to illustrate how the four base data interfaces can be combined to yield data services with different functionalities.

Fixed-format file (DataAccess). This particularly simple data service provides access to a single data virtualization that is

??? (DataDescription, DataAccess). DataDescription to expose SDEs, DataAccess to access file.

Directory (DataDescription). DataDescription to expose SDEs for files, which include GSHs for data services for individual files. (?)

Database (DataDescription, DataManagement) "Provision" some new tables in an existing DBMS instance. Examine the tables that are already there. Create a new set of tables.

Database (DataDescription,DataManagement, DataAcess) Provision a new system Install the DBMS, use DataManagement to start the DBMS, and then to create the database and tables, use DataAccess to create the content.

Examples to be provided.

7 Contributors

We gratefully acknowledge the contributions made to this document by the following people:

Ann Chervenak, Karl Czajkowski, Carl Kesselman, Allen Luniewski, Cecile Madsen, Susan Malaika, Inderpal Narang, Michael Swanson

8 Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we've missed) Malcolm Atkinson, Jeff Frey, Steve Graham, Simon Laws, Norman Paton, and David Pearson.

This work was supported in part by IBM; by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S.

9 Issues

This section is a grab-bag of issues that we believe need to be further discussed and clarified, either here or elsewhere.

DAIS Mappings:

- DAIS SDEs for DRM and DR (Data Resource Manager and Data Resource) → DAI: DataDescription – RelationalDescription, XMLDescription

- DAIS DAS (Data Access Session) and Data Request → DAI:DataAccess

- DAIS DataSet: → DAI: Data Set

Caches and replicas. How do caches fit? How do replicas fit?

Data placement. Where does that go?

Service granularity. We discuss in Section 2.2.2 the advantages of using GSHs as a universal global naming scheme for any data element that needs to be visible externally to its data source. This strategy implies that the creation of a new data service needs to be an extremely lightweight operation.

Metadata. Services?

Provenance. Can we—and, if so, how do we—build on these mechanisms to define data services that maintain and provide access to provenance information.

10 References

1.   Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S. and Xu, M. Agreement-Based Grid Service Management (OGSI-Agreement). Global Grid Forum, Draft, 2003.
2.   Czajkowski, K., Foster, I., Sander, V., Kesselman, C. and Tuecke, S., SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. *8th Workshop on Job Scheduling Strategies for Parallel Processing*, Edinburgh, Scotland, 2002.
3.   Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, *35* (6). 37-46. 2002.
4.   Paton, N.W., Atkinson, M.P., Dialani, V., Pearson, D., Storey, T. and Watson, P. Database Access and Integration Services on the Grid. U.K. National eScience Center, 2002. www.nesc.ac.uk.
5.   Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maguire, T., Sandholm, T., Snelling, D. and Vanderbilt, P. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum, Proposed Recommendation GFD-R-P.15, 2003. www.ggf.org.