GWD-I (draft-ggf-dais-dataservices-03)
Data Access and Integration Services (DAIS)
http://forge.ggf.org/projects/dais-wg

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

February 24, 2004

OGSA Data Services

Status of this Memo

This document provides information to the community regarding the specification of OGSA-compliant data services. Distribution of this document is unlimited**.** This is a DRAFT document and continues to be revised.

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 web service that implements one or more of three base data interfaces to enable access to, and management of, data resources in a distributed environment. Data services are built on the (OGSI-derived) pattern of modeling stateful resources as web services, the WS-Resource framework. The base data interfaces, DataAccess, DataFactory, and DataManagement, define basic 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 service-oriented architecture.

GLOBAL GRID FORUM
office@gridforum.org
www.ggf.org

Table of Contents

1 Introduction

A service-oriented representation of data can allow data to be treated in the same way as other resources within the Web 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 [4] when manipulating data. For example, we can use lifetime management mechanisms provided by the Open Grid Services Infrastructure—or, as assumed here, by its proposed evolution, the WS-Resource framework [3]—and represent agreements concerning data access via WS-Agreement [1].

The design of appropriate interfaces and behaviors for such "data services" is made complicated by the heterogeneous data resources and data access methods encountered 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 resources. 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. Heterogeneity in data access semantics leads to additional challenges. For example, having defined an interface that makes the result of a query over a SQLAccess service accessible as a RowSet, we must be concerned with whether this RowSet is materialized or derived on demand and, if derived, what consistency guarantees are provided. Does an update on the RowSet write back to the database, or is it local?

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 resources. ("Data virtualization" is one of several 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) Working Group [6].

In our service-oriented treatment of data, a data virtualization is represented by, and encapsulated in, a *data service*, a web service backed by a WS-Resource that has a *WS-Resource Properties document* that describe key parameters of the virtualization, and with *operations* that allow clients to access the data using appropriate operations, derive new data services from old, and/or manage the data service. (We use the term *resource properties* to refer to the WS-Resource Properties document associated with a data service's WS-Resource.)

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 a WS-Resource Properties document that defines 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) accessible via the web service's property document, 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 resource(s).

[1] In the rest of this document we will use the term *resource properties* to refer to the WS-Resource Properties document associated with a Data Service's WS-Resource.

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 three *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 WS-Resource compliant Web service that implements one or more of these base data interfaces or which provides a WS-Resource Properties document as described later in this document. (We use the phrase "WS-Resource compliant" to mean a web service that follows the WS-Resource framework approach [3].).

The three base data interfaces are as follows. We show below how these base interfaces can be combined and extended to define various interesting services. (In this document the phrase "extends", or "extends an interface" means that the new interface is created by a process of aggregation of interfaces.)

- *DataAccess* provides operations to access and/or modify the contents of the data virtualization encapsulated by the data service.
- *DataFactory* provides operations to create a new data service with a data virtualization derived from the data virtualization of the parent data service
- *DataManagement* provides operations to monitor and manage the data service's data virtualization, including (depending on the implementation) the data resources (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 Web services resource framework (WSRF) interfaces but also WS-Agreement interfaces [1], which are used to incorporate agreements (e.g., quality of service guarantees, or payment information) into the various data operations. (This document is written in terms of WS-Agreement. However, WS-Agreement is an in–progress specification as of this writing. We discuss below how a data service will be specified until WS-Agreement is finalized.)

Relationship management functionality will also be important, as a means of representing and managing relationships among virtualizations, such as multiple virtualizations against the same data resource, and dependencies between virtualizations. DAIS may either build on OGSA relationship management services here or, given that those services have not yet been defined, develop DAIS-specific functions that can perhaps be used as a use case by those developing an OGSA relationship proposal.

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 three 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	A (virtual) file system, JPEG file, relational database, column of a relational table, random number generator.	§2.1

[1] 3. Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Storey, T. and Weerawaranna, S. Modeling Stateful Resources with Web Services. Globus Alliance, 2004. .

;
[2]

	by a data service.	number generator.	
Base data interface	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	A WS-Resource compliant Web service that implements, via its underlying WS-Resource resource, one or more of the three 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. A data service provides access to a single 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 resource	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, program or another data service.	§2.1
Resource manager	The logic that brokers requests to underlying data resource(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

[1] In this document the phrase "extends", or "extends an interface" means that the new interface is created by a process of aggregation of interfaces.

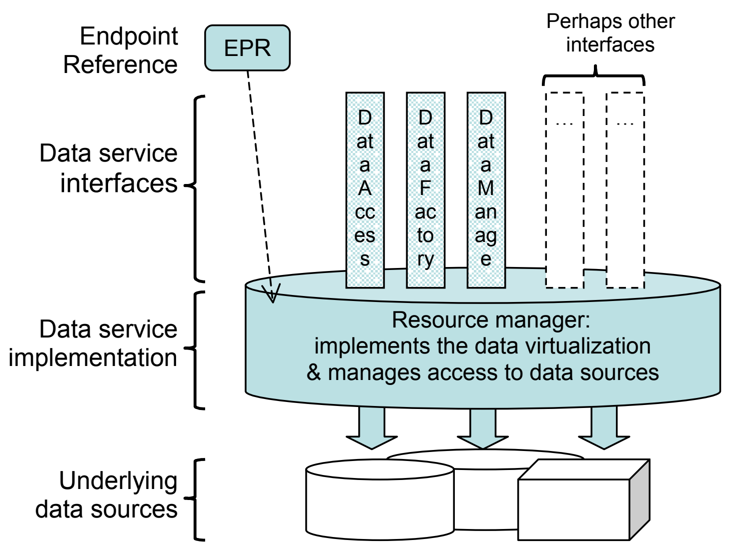

Figure 1: Architecture and scope of the OGSA data service concept. The shaded areas denote a data service, the three base data interfaces, and an endpoint reference that references the data service. The service's implementation (sometimes referred to as a "resource manager") brokers requests to underlying data resource(s), via the service's data virtualization, through the data interfaces.

2 Data Virtualizations

The data virtualization concept 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 resource* to denote a system- or implementation-specific physical or logical construct that provides access to data. Examples of a data resource include a file, file system, directory, catalog, relational database, relational table, XML document, and 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 resources. A data service can itself be a data resource for another data service.

While different data resources have their own peculiarities, service-oriented interfaces can be defined and implemented that make any particular data resource 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 a data service that makes 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 data resource'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 resources. The abstraction that a data virtualization provides of its underlying data can be simple (e.g., a straightforward service-oriented rendering of the underlying data resource's interface) or complex (e.g., a transformation from files to tables); may correspond to a subset of an individual data resource (e.g., a view on a database or a file within a file system) or federate multiple data resources; and can involve simple data access or computational transformations of underlying data. A data virtualization is realized via a data service.

Mappings between data virtualizations and data resources 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 resource is virtualized simultaneously at different levels of granularity (see Figure 2). For example, we might define data virtualizations for an entire file system (with associated operations for managing the file names and metadata); for 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 for 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 resource(s) 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 resources. 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.

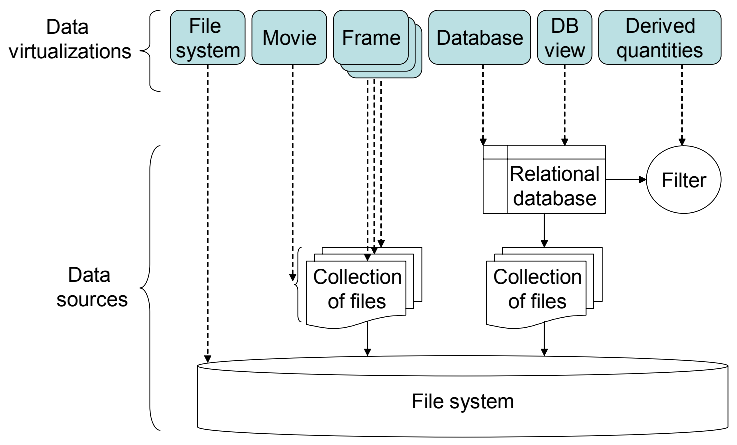

Figure 2: An illustration of how different data virtualizations can provide different views of the same or different parts of a data resource. In the OGSA Data Services framework, each of these virtualizations will be implemented by a distinct data service.

2.2 Representing Data Virtualizations

As noted above, a data virtualization is represented by a *data service*, WS-Resource compliant Web service that implements one or more of the base data interfaces and/or a WS-Resource Properties document. There is a one-to-one relationship between data services and data virtualizations: a data virtualization is simply the state of a data service as projected through the service's interfaces.

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

2.2.1 Resource Properties to Represent Data Service State

We use the resource properties 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. The resource properties may also be used to describe other metadata, such as who produced the data, its purpose, and abstract identifiers and properties of portions of the data. This use of resource properties enables inspection and discovery via standard mechanisms. We anticipate that the resource properties used within various specific domains will be standardized within GGF, OASIS, DMTF, and/or other (perhaps discipline-specific) standards bodies.

2.2.2 Naming Data Services

WS-Resource compliance also means that each data service (and thus the data virtualization associated with that service) has a name (its WS-Addressing endpoint reference) that is used to direct requests to that particular data service. See [3] for a detailed discussion of naming of a stateful resource such as a data service. Note that the endpoint reference to a data service does not, in general, provide a globally unique name for data. Rather, it only provides a means to denote a particular data service.

2.2.3 Lifetime Management of Data Services and Resources

A data service follows the WSRF lifetime management use case patterns [3]. These patterns can be used to manage the lifetime of data services and also perhaps, depending on context, their underlying data resource(s).

While some data services will be created via out-of-band mechanisms and will map to persistent data resources (see Section 4), others can be created dynamically by data factories, and/or have explicitly managed, finite lifetimes. That is, an initial lifetime may be established at the time of creating the data service (and may be modified later) and/or the service may be destroyed via explicit, data-service-specific, 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 resource(s) persists; in others, the lifetime of the data service and the underlying data resource(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. The behavior adopted in a particular instance may be a property of a service or, alternatively, may be specified by policy associated with a particular data resource and/or customer business environment. Just how such behaviors are specified and enforced will presumably be an important issue to discuss and standardize, for example within the context of an operational data center.

2.2.4 Representing Sessions as Transient Services

A client negotiates an agreement granting access to a data service. The data service may then create a transient service (i.e., a service with limited lifetime) to represent and manage the client's context or session. This context may include not only access control and choice of data content access protocol, but also identification of data content to be accessed. Thus, a data virtualization is needed as a starting point.

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 can 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 web service, be accessed by many clients and, like any web 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 service (and thus its data virtualization) to its underlying data resource(s) is determined by the data service's WS-Resource[1]. A data resource 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 resource to create and manipulate data virtualizations of a data resource.

Not withstanding the statements made in the first paragraph, a data resource 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 resources encapsulated by the data service to provide a "virtualized" instance of that data as part of the Web 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 resource. 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.

[1] We will use the phrase "Data service implementation" to mean the WS-Resource that provides the data context for the data service web service.

3 Data Service Compliance

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

Each of the three *base data interfaces*—DataAccess, DataFactory, and DataManagement—defines operations that can be invoked against a data service. As illustrated in Figure 3, each base data interface extends a WS-Agreement interface. 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 DataAccess might include RelationalAccess, WebRowSetAccess, FileSystemAccess, and FileAccess.

A data service may implement various combinations of these requirements. For example, a simple data service that virtualizes a file might just provide a FileProperties resource properties that describes the file, along with a FileAccess interface for reading and writing sections of the file. A more complex data service might support multiple extensions to the base 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 FileProperties resource properties and the FileAccess interface, the data service might also implement a JPEGProperties resource properties and a JPEGAccess interface to allow more specialized description and access. As a second example, a complex data service might provide both RelationalAccess and XMLAccess access to its data, with the corresponding RelationalProperties and XMLProperties resource properties provided as its resource properties.

In the rest of this section, we describe data service resource properties and the three 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.

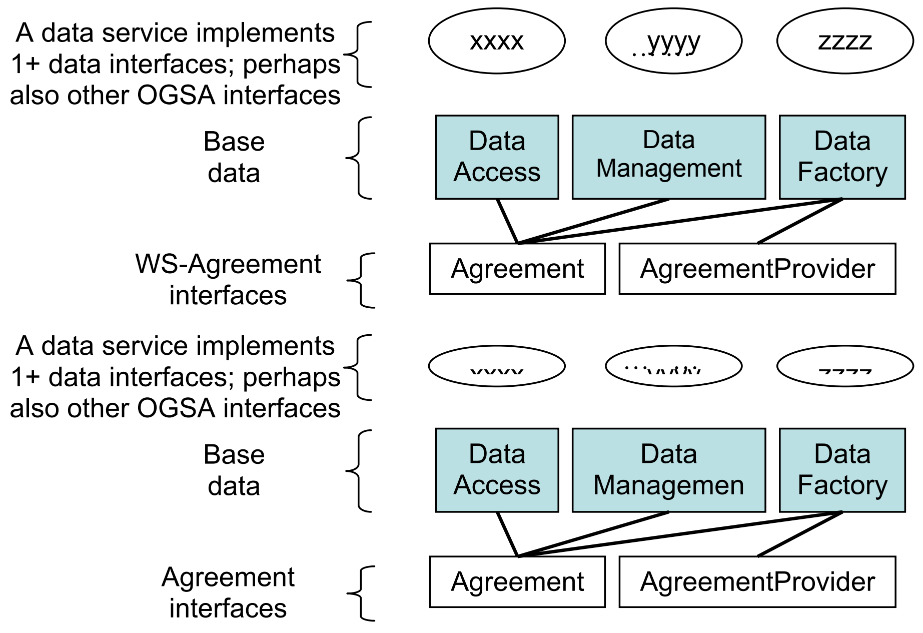

Figure 3: Data services (at top) implement one or more of the base Data interfaces (shaded) or extended versions of those interfaces. The Data interfaces themselves extend WSRF and WS-Agreement interfaces. (Lines linking interface names represent interface extension.)

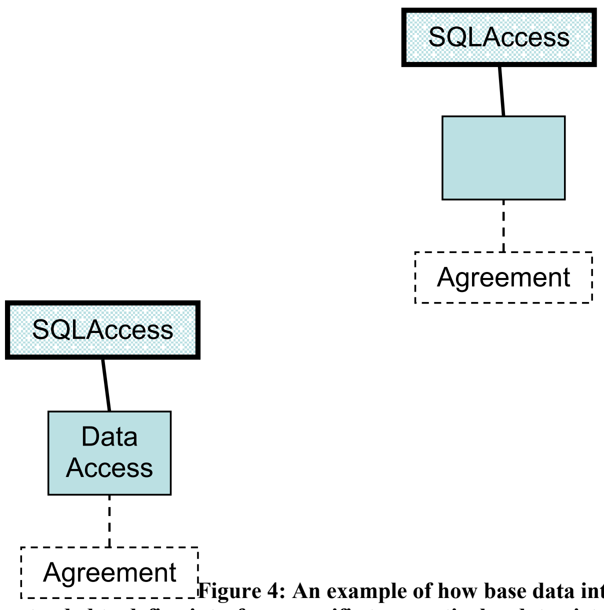

Figure 4: An example of how base data interfaces (in this case DataAccess) can be extended to define interfaces specific to a particular data virtualization. The unshaded dashed box denotes non-data service interfaces and the shaded boxes denote data service interfaces.

Table 2: Synopsis of the base data interfaces.

Interface	**Extends**	**Resource properties (in addition to that in extended interface)**	**Operations (in addition to those in extended interface)**
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 creation operation for configuring the derived data virtualization	None. (The creation operations are data service specific.)
Data Management	Agreement	Information about the data virtualization, for example for performance monitoring.	Configuration of the data service.

3.1 Data Service Properties Document

The WS-Resource Properties document of a data service, its resource properties, defines the XML elements that describe the data virtualization supported by a particular data service. These 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.

The base resource properties includes all elements (to be defined in future work) that are assumed to be common across all data virtualizations. This includes all information that is needed to support the three base interfaces to a data service. More specialized interfaces may extend the base resource properties to introduce XML elements that are relevant to specific data virtualizations, for example defining a data virtualization's schema along with generic properties and associated policies. Extensions to the base resource properties may be generic (e.g., a set of RelationalProperties elements that describe database names, table names, column names, and column types) or domain-specific (e.g., a climate modeling or financial analysis data virtualization).

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

RelationalProperties defines elements 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).

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

XMLCollectionProperties defines elements that describe a collection of XML documents, including their XML Schemas and number of documents.

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

FileProperties defines elements 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 WS-Agreement. The Agreement interface allows for inspection, monitoring, and possibly re-negotiation of the agreement terms governing use of this data service.

The base DataAccess interface introduces operations (to be defined in future work) that are common across all data virtualizations. Extensions to this interface contain operations that are specialized for access to particular types of data virtualizations.

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 a 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 RelationalProperties extensions to the data service resource properties. 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 RowSetProperties and CursorRowSetAccess interfaces.

CursorRowSetAccess: Cursor based access to a row set data virtualization. This interface will often be used in conjunction with the RowSetProperties extensions to the Data service resource properties. 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 XMLCollectionProperties extensions to the data service resource properties. 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 FileProperties extensions to the data service resource properties. Like CursorRowSetAccess, this is a "stateful" interface that will typically reside on a data service that is created by a factory operation (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 FileProperties extensions to the data service resource properties. Like CursorRowSetAccess, this is a "stateful" interface that will typically reside on a data service that is created by a factory operation (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 FileProperties and FileSystemProperties extensions to the data service resource properties. 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 WSRF-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 WS-Resource pattern does not define explicit factory operations. Instead it provides general guidelines for creation of a WS-Resource. The DataFactory interface is in the spirit of these guidelines and supports a request to create a new data service whose data virtualization may derive 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. The new data service may map only to its parent's data resource(s); alternatively, the creation of the new data service may also involve the creation of content to be contained in a data resource distinct from the parent data resource(s). A data factory can also create an "empty" data service (i.e., one that does not contain any data), perhaps to receive incoming data.

The base DataFactory interface introduces operations (to be defined in future work) that are common across all data services. Extensions to the base DataFactory interface can define factory parameterizations specialized for derivations from and to particular data services.

The DataFactory interface extends the WS-Agreement AgreementProvider interface. A request to a DataFactory results in the creation of a new service, and the return of an endpoint reference to that new service to the requestor. This endpoint reference can then be used by requestors to direct requests at any operation implemented by the new service, including the various data service interfaces. New data service instances created by a DataFactory have their own lifetime, service data, and state, and may be relatively transient or long-lived, as defined by the DataFactory operation used to create the new data service.

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 service. Each data service (and thus the data virtualization it encapsulates) has, by virtue of being represented as a Grid service, a name (its GSH) that is globally unique for all time an endpoint reference that denotes that particular data service. Access interfaces may be used to produce and consume data sets related to, or derived, from this data service, using parameters that are meaningful only within the context of operations against that particular data service. But a client may want to obtain a more portable name for a particular derivation of a data service, for example, to give to another client. In this situation, a DataFactory operation should be used to create a new "derived data service" as a separate data service, so that the derived data service has a name (its endpoint reference), and can be managed using standard WSRF mechanisms. Examples of such derived data services include creating a data service that has as its data virtualization 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 resource 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 a data resource. 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 factory operation should be used to create a new data service 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 service. 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 factory operation that adds a new file to the file system, and creates a data service 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 operation for creation of a data service are defined using the Agreement language defined by WS-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. WS-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 WS-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. 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. The SQLFactory interface on this data service would allow for the creation of a

new data service whose data virtualization is an (arbitrary) relational subset or view of the parent's relational database, and that supports the RelationalProperties resource properties and appropriate SQLAccess and SQLFactory interfaces. The SQLFactory interface could also allow for the creation of a new data service whose data virtualization is a result set from a single SQL select command, and that supports the RowSetProperties resource properties extension 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 WSRF resource properties 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 resources 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 resources, with the appropriate specialized resource properties and DataAccess interface on the new data service. An extension of this interface would allow for creation of federations of the underlying resources.

We emphasize again that the creation of a data service via a factory operation does *not* necessarily imply materialization of the new data virtualization. The new data service may just be another virtualization over the same underlying data resource(s). For example, creating a file virtualization using the FileSelectionFactory, or creating a view on a relational database using the SQLFactory, does not require (nor does it 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 interface 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 Resources

The DataFactory interface only allows for the derivation of new data services 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 operation might first be used to create a new federated data service with an empty relational data virtualization, or a derivation of one of the input relational data services. 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 service, and thus its data virtualization—and, indirectly, its underlying data resource(s). The base DataManagement interface introduces operations (to be defined in future work) that are common across all data services. Extensions to this interface may introduce operations that are specialized for management of particular types of data service.

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

The DataManagement interface further may provide operations for configuring the base access policies of a data service, thus bounding the agreements that requestors may negotiate with the data services

Various extended DataManagement interfaces may affect only the data virtualization implemented by the data service, or they may additionally affect the underlying data resource. For example, a RelationalDatabaseManagement interface implemented by a relational DBMS may allow both the definition of data services from the underlying data resource (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 resource (for example, add new tables to a database in the DBMS).

Management interfaces are out of scope of the 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.

3.5 Data Service Considerations

The data service concept provides a general framework for data virtualization. However, we expect that in practice a data service will impose constraints on its data virtualization, including the following:

- It must be possible to specify the data to be virtualized as part of some factory operation.
- It must be possible to describe all virtualized data via its resource properties.
- The DataAccess operations must have the expressive power to provide access to all of the virtualized data.

The need for consistency among these three sets of operations will impose constraints on the specifics of the data service's resource properties, the factory operations (including those in the DataFactory interface) and DataAccess interfaces. For example, if a factory operation creates a data service that supports relational access, then the relational schema information available via the its resource properties should provide enough information to denote all of the data that has been virtualized. Moreover, all of that data should be accessible via a SQL-based DataAccess operation.

Some data services will provide more than one type of virtualization of their data (e.g., XML and relational). This document takes no position on the necessity of consistency in either the resource properties or the DataAccess interface between these distinct types of virtualizations.

Nothing in this document should be construed as inherently limiting the complexity or size of a data virtualization. In fact, we fully expect that there will be extremely complicated data virtualizations. This section is intended simply to observe that data virtualizations must meet some minimal consistency requirements in the data service interfaces.

4 Root Data Services

The abstractions and interfaces that we have described allow for the creation, destruction, composition, and federation of data service. However, just as some WS-Resource compliant web services are created and destroyed via mechanisms other than WSRF 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 all or part of the underlying data resource into one or more data virtualizations, each wrapped by a data service. The handles (end point references) 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 WS-Agreement

The draft WS-Agreement specification [1] is based upon the OGSI 1.0 specifications and builds on concepts introduced within the Service Negotiation and Access Protocol [2] and WSLA [5] 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. WS-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.

WS-Agreement is a critical foundation to the data services described here. Access to all data services is governed by agreements, as defined by WS-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 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 WS-Agreement base term type, and which can be combined in a single WS-Agreement defined agreement element. Terms may be general or specific to an extended DataFactory, and may address, for example, how the new data service should be derived from the parent (factory) data service; 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 three 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

??? (DataAccess). The resource properties exposes file properties while DataAccess is used to access the file.

Directory (resource properties only). The resource properties exposes properties of files, which include endpoint references for data services for individual files. (?)

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

Database (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 Timing Issues

This data service concept has dependencies on a number of other ongoing standards efforts. This document has been written to reflect a final state in which all of these have been finalized. Unfortunately, as of this writing (January, 2004), the OGSA Data Services work will be completed before some other standards are ready. This section is intended to discuss the resulting timing issues and their impact on intermediate states of the realization of this document.

The Data Services concept depends intimately upon the WS-Resource framework standards. This dependency is inherent so the Data Services standard can complete no earlier than the WS-Resource framework standards.

As discussed in Section 5, OGSA Data Services are also data services is dependent upon WS-Agreement. However, the current expectation is that WS-Agreement will finalize much later than the anticipated completion date of Data Services. Pending completion of the WS-Agreement specification, we suggest that the DAIS working group create an interim, DAIS "private", version of WS-Agreement that will be used in the final results of the DAIS working group. We further suggest that this version be chosen in a way that meets a number of goals. It should be chosen so that there is a reasonable expectation that it will end up being a subset of the ultimate WS-Agreement specification. It should be chosen to be as simple as is reasonable given the goals for Data Service. Finally, it should be chosen to minimize the possibility of rework by data service implementers and by clients of data services. We suggest that areas for simplification include minimizing the number of agreement terms defined in the base data services specification and eliminating the option of negotiating and renegotiating agreement terms.

Section 1 suggests the need for a relationship manager to track the relationship between different data services, especially those derived from other data services. We do not believe that a relationship manager will have any impact on the data services specification so the timing of these two specifications can be independent of one another.

8 Conclusions

We have presented a general framework for including data resources into the service-oriented Open Grid Services Architecture (OGSA). We define an OGSA *data service*, a WS-Resource compliant web service that implements one or more of three base data interfaces to enable access to, and management of, data resources in a distributed environment. These base interfaces, DataAccess, DataFactory, and DataManagement, build on WSRF mechanisms to define basic service data and/or operations for representing, accessing, creating, and managing data services. Specific 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 service-oriented architecture.

This initial presentation needs to be extended in various ways to provide a complete description of this data services framework. In particular, we need to develop specifications for the syntax (WS-Resource Properties document and operations) and semantics of the three base data interfaces. We also want to develop principles for designing services using the framework.

9 Contributors

We gratefully acknowledge the contributions made to this document by Ann Chervenak, Karl Czajkowski, Carl Kesselman, Allen Luniewski, Cecile Madsen, Susan Malaika, Inderpal Narang, and Michael Swanson.

10 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. Department of Energy, under Contract W-31-109-Eng-38 and DE-AC03-76SF0098; by the National Science Foundation; and by the NASA Information Power Grid project.

11 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. Endpoint references are used as the universal naming scheme for any data element that needs to be visible externally to its data resource. 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.

12 Summary of Changes

This section provides a brief summary of the changes made in this version of the document.

- References to concepts from OGSI 1.0 such as grid service and GSH were replaced by terms such as endpoint reference and WS-Resource.
- The DataDescription interface was replaced by a data service WS-Resource Properties document. Examples were changed to reflect the change from "Description" to "Properties document".
- Section 7, Timing Issues, was added.
- Small changes were made in the use of the term "data virtualization" to minimize the confusion that this term could cause.
- Figures were updated to reflect the above changes.

References

1. Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S. and Xu, M. Agreement-Based Grid Service Management (WS-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., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Storey, T. and Weerawaranna, S. Modeling Stateful Resources with Web Services. Globus Alliance, 2004.
4. Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, *35* (6). 37-46. 2002.
5. Keller, A. and Ludwig, H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, *11* (1). 2003.
6. 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.