Mario Antonioletti, University of Edinburgh
Malcolm Atkinson, National e-Science Centre
Neil P Chue Hong, University of Edinburgh
Amy Krause, University of Edinburgh
Susan Malaika, IBM
Gavin McCance, University of Glasgow
Simon Laws, IBM
James Magowan, IBM
Norman W Paton, University of Manchester
Greg Riccardi, Florida State University

GGF Database Access and Integration Services Working Group
Category: INFORMATIONAL 6th June 2003

Grid Data Service Specification

Status of This Memo

This memo provides information to the Grid community regarding the specification of Grid Database Services. The specification is presently a draft for discussion. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

Data management systems are central to many applications across multiple domains, and play a significant role in many others. Web services provide implementation neutral facilities for describing, invoking and orchestrating collections of networked resources. The Open Grid Services Architecture (OGSA) extends Web Services with consistent interfaces for creating, managing and exchanging information among Grid Services, which are dynamic computational artifacts cast as Web Services. Both Web and Grid service communities stand to benefit from the provision of consistent, agreed service interfaces to database management systems. Such interfaces must support the description and use of database systems using Web Service standards, taking account of the design conventions and mandatory features of Grid Services. This document presents a specification for a collection of grid data access services. The proposal is presented for discussion within the Global Grid Forum (GGF) Database Access and Integration Services (DAIS) Working Group, with a view to the document evolving to become a proposed recommendation. There are several respects in which the current proposal is incomplete, but it is hoped that the material included is sufficient to allow an informed discussion to take place concerning both its form and substance.

Contents

Mario Antonioletti, University of Edinburgh
Malcolm Atkinson, National e-Science Centre
Neil P Chue Hong, University of Edinburgh
Amy Krause, University of Edinburgh
Susan Malaika, IBM
Gavin McCance, University of Glasgow
Simon Laws, IBM
James Magowan, IBM
Norman W Paton, University of Manchester
Greg Riccardi, Florida State University

GGF Database Access and Integration Services Working Group
Category: INFORMATIONAL 6th June 2003

1. Introduction

This document presents a specification for a collection of grid data access services. The proposal is not *ab initio*, in that the following documents (at least) have been important in shaping our understanding of such services [Atkinson 02, Bell 02, Collins 02, Krause 02, Paton 02, Raman 02]. The proposal presented here has also evolved from earlier versions of this document, in the light of discussions within the DAIS Working Group of the Global Grid Forum. In particular, in the time since GGF7, the introduction of a new conceptual model for the data access services has led to a significant reorganization of the specification.

The following principles have guided the development of the specification.

1. The specification is intended to provide service-based access to *existing* data management systems. As such, it is assumed that there is no such thing as a Grid Database System, but rather that existing databases require the provision of certain middleware components to make them available in a Grid or Web Services setting.
2. As there are several widely used data management paradigms (e.g., relational, object, XML), the specification seeks to accommodate these within a consistent framework. Thus those aspects of a service interface that are independent of the underlying data model are shared by services that support the different paradigms. For example, it is held that result delivery facilities and transaction models are essentially orthogonal to the kind of database being accessed. The specification presented here covers relational and XML databases, but is intended to be extensible for use with other storage paradigms.
3. A characteristic of Web and Grid Services is that metadata is important. In the specification, it is assumed that a service must provide sufficient information about itself to allow the service to be used given the specification of the service and the metadata provided by the service. Service metadata is made available using Service Data Elements [Tuecke 03].
4. Data access services should peacefully coexist with other Web and Grid Service standards. As such, the specification adopts XML Schema for describing structured data and WSDL for describing service interfaces. Furthermore, the specification seeks to stay clear of issues covered by other Web and Grid Service specifications (e.g., it is largely silent on transactions, assuming that the WS-Transaction proposal [Cabrera 02-b] is, or will evolve to be, sufficient and appropriate for characterizing the transactional behavior of distributed services). Overall, the proposal seeks to accommodate the distinctive features of individual systems, while easing data access and integration activities within a Grid setting.
5. The specification should be orthogonal to Grid authentication and authorization mechanisms. These mechanisms are required to some extent by all Grid Services, so we rely on them being defined in a more general context. Many data management products define fine-grained access to the data they hold, for example, down to the column and row level for relational databases. A data access service implementation can choose the level of authorization granularity exposed to the Grid users; the specification does not seek to mandate this.
6. The specification is defined semi-formally. That is, the syntax of the specification is presented formally, as WSDL and XML Schema documents, whereas the semantics of these specifications is provided only informally in the accompanying text.
7. The specification seeks to support higher-level information-integration and federation services. As such, features such as service description and data delivery support, have been designed with a view to providing the necessary ‘hooks’ for the developers of such services.

Some familiarity with XML Schema [Fallside 01], WSDL [Christensen 01] and the Open Grid Services Infrastructure (OGSI) [Tuecke 03] is assumed in what follows. This document is written in the context of Version 1.0 (March 13th 2003) of the OGSI Specification.

2. Notational Conventions

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" are to be interpreted as described in RFC-2119 [Bradner 97].

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

Prefix	Namespace
dais	http://www.ggf.org/namespaces/2003/06/DAIS
ogsi	http://www.ggf.org/namespaces/2003/03/OGSI
gwsdl	http://www.ggf.org/namespaces/2003/03/gridWSDLExtensions
sd	http://www.ggf.org/namespaces/2003/02/serviceData
wsdl	http://schemas.xmlsoap.org/wsdl/
http	http://www.w3.org/2002/06/wsdl/http
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
webr	http://java.sun.com/xml/ns/jdbc/ (contents subject to change)
sqlxml	http://standards.iso.org/iso/9075/2002/12/ (temporary location)

3. The Model

This section describes the principal concepts of relevance to the specification, outlines the way in which these concepts are realized in the specification through service data and operations, and indicates how the specification relates to wider issues in the grid and web services space.

3.1 The Conceptual Model

This subsection introduces concepts and terminology that are used throughout the specification. As the model is conceptual in nature, items introduced in this section may not map directly to implementation constructs, and in fact, it is possible to configure components from the specification in ways other than those discussed in this section. However, the model presented here is paradigmatic in the sense that it is representative of, and has served to motivate, many aspects of the specification provided in later sections.

3.1.1 Introduction

The conceptual model captures data management systems, the data resources they contain, and the data sets resulting from data requests performed on these data resources.

The following principles are observed:

- DAIS offers a way of exposing native languages and/or APIs of resource managers through grid services.
- DAIS does not attempt to hide or interpret the underlying data model of the accessed resources nor does it hide the access languages supported by the resource managers.
- DAIS, as much as possible, tries not to inhibit access to mechanisms supported by the accessed resource managers

The term "external" is used to identify resources that exist outside of the OGSI compliant grid. A vocabulary is defined for external resources of interest. For example, consider a relational database management system, as illustrated in Figure 1. An *external data resource manager* is an installed instance of a data management system, such as Oracle 9i or DB2. Such a system may manage zero or more *external data resource*s, each of which is a distinct database over which requests may be phrased. A collection of data that is not being explicitly managed by an external resource manager, such as the result of a query evaluated over an external data resource, can be referred to as an *external data set*.

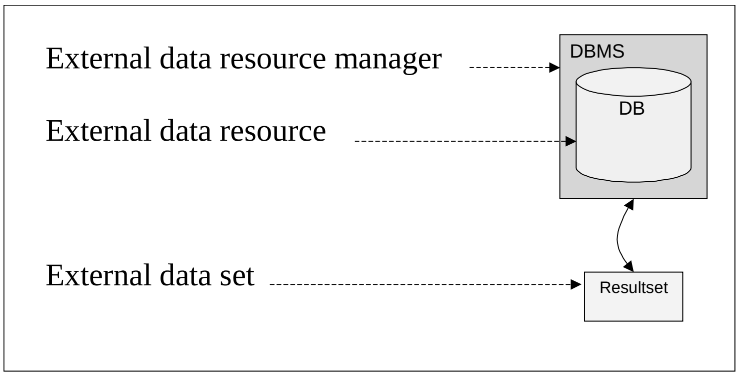

Figure 1 - External Resources

A vocabulary is defined for a logical representation of these external resources and the requests that are applied to them. Again, taking a relational database management system as an example, Figure 2 illustrates concepts that may be created or used by a requester making use of the external resources from Figure 1. A *data resource manager* is a logical representation of an external data resource manager, and a *data resource* is a logical representation of an external data resource. Both a *data resource manager* and a *data resource* could, for example, be implemented as OGSI services. A requester interacts with a data resource by way of a *data activity session*, which provides a context and manages state for a collection of *data requests* addressed to the data resource. The evaluation of a request may involve access to or creation of one or more *data sets*, which in turn manage access to external data sets.

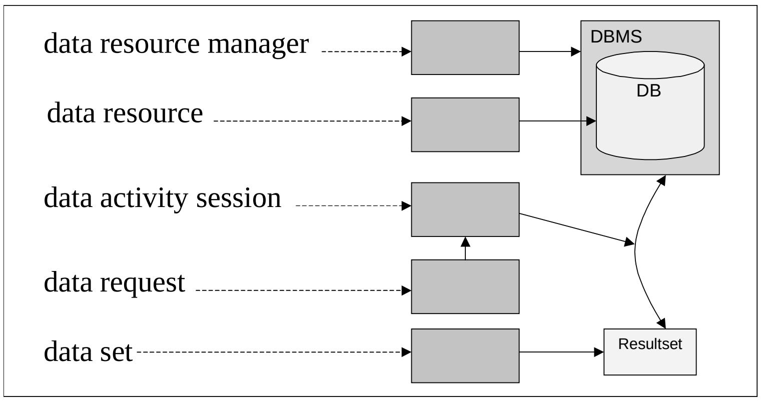

Figure 2 - Logical Resources

The DAIS specification maps the logical resources of the conceptual model onto OGSI [Tuecke 03] compatible portTypes, which describe their interface.

3.1.2 External Data Resource Manager

In the *external* world there is a universe of *external data resource managers:* EDRM[1]. Each edrm (edrm $\in$ EDRM) represents an installed instance of a system designed to manage data, such as

[1] Sets are given upper-case names and their members normally have the same spelling in lower case.

a database management system. Figure 3 illustrates the collection of external data resource managers.

Each edrm is installed, managed and eventually removed in accordance with its defined operational procedures. Management of the edrm itself is out of scope of the DAIS specification.

Each edrm exposes interfaces for management of the “external data resources” that it contains. These interfaces are specific to each type of edrm. For example, a file management system manages files organized into directories. A database management system manages databases.

In order that an edrm can be used there must be a means of identifying that edrm, c.f. the url mechanisms used to locate web services, file systems and database management systems. We refer to this as an edrm_name.

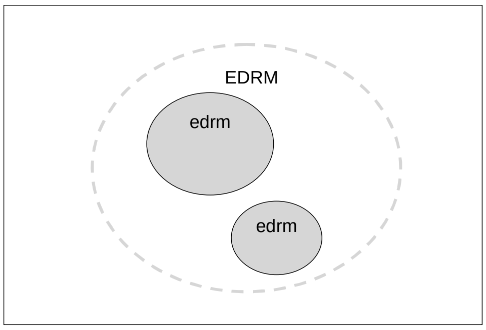

Figure 3 - The Set Of External Data Resource Managers

3.1.3 External Data Resource

There is a universe of *external data resources*: EDR. Each edr (edr $\in$ EDR) is managed by an edrm, as illustrated in Figure 4. The structure and semantics of an edr are dependent on the edrm that manages it. For example, a relational database, managed by a relational database management system. The existence, creation and termination of edr, their operation policies and available functions are determined by decisions, standards and mechanisms, which are normally not specified by DAIS.

In order that an edr may be used there must be a means of identifying each edr, c.f. the database naming schemes in JDBC and ODBC. We refer to this as an edr_name.

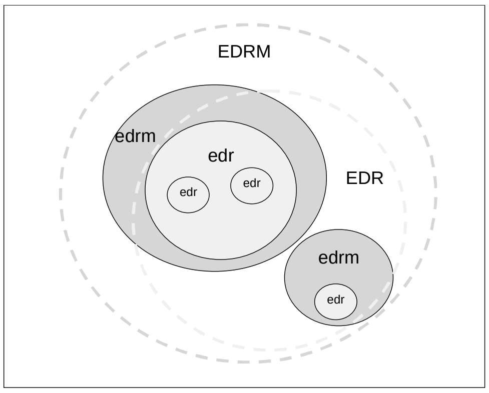

Figure 4 - The Set Of External Data Resources

Relationships:

- An edrm manages zero or more edr.

- An edr is managed by one edrm.

3.1.4 External Data Set

There is a universe of *external data sets*: EDS. Each eds (eds $\in$ EDS) represents a collection of data that is not explicitly linked with an edrm. This is data that is generated by an application before being committed to an edr, such as a sequence of calculated values or data that is extracted from an edr, such as a relational result set.

Typically data is introduced to or extracted from an eds using a limited set of operations that vary depending on the type of data that the eds represents. An eds representing relational results may allow tuples to be extracted individually. An eds representing an XML sequence may allow xml fragments to be extracted.

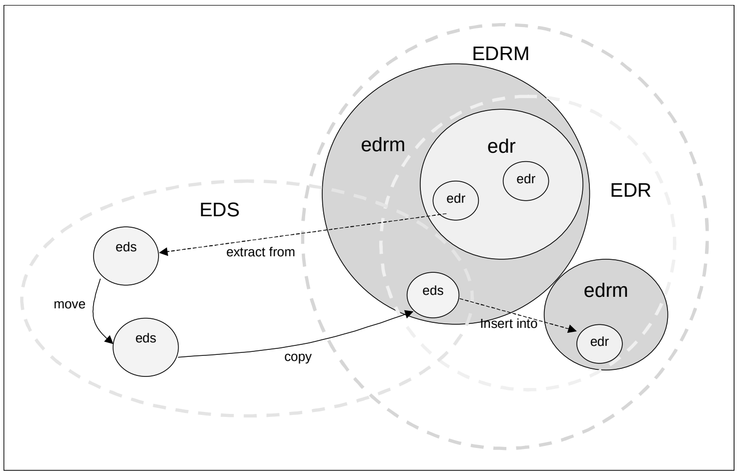

Figure 5 - The Set Of External Data Sets

In many scenarios where data is introduced directly into an edr, an eds is not represented as a service in its own right, for example, an application submitting SQL insert statements to a edr that is a database can include the data alongside the SQL insert statement.

Relationships:

- An eds represents a free standing set of data, and hence no relationships between eds are defined.

3.1.5 Data Resource Manager

There is a universe of *Data Resource Managers* (DRM). Each drm (drm $\in$ DRM) is a grid service and has the usual properties of a grid service specified by the OGSI specification. Hence, identification of a drm is achieved with grid service handles and grid service references. Registries may offer other means of identifying drm and provide a mapping to their handles.

The drm exposes management interfaces for the edrm and provides interfaces for managing the edr.

A drm is created and inserted into the set DRM by a factory or some out of band mechanism not defined by DAIS. On creation, the drm is bound to edrm using the edrm_name, as illustrated in Figure 6. This edrm must already exist.

The lifetime of a drm is managed using the usual OGSI mechanisms. The lifetime of the edrm is unaffected by termination of the drm. Conversely, if the edrm is terminated by external forces, the drm remains extant for its destined life and issues appropriate error responses and notifications.

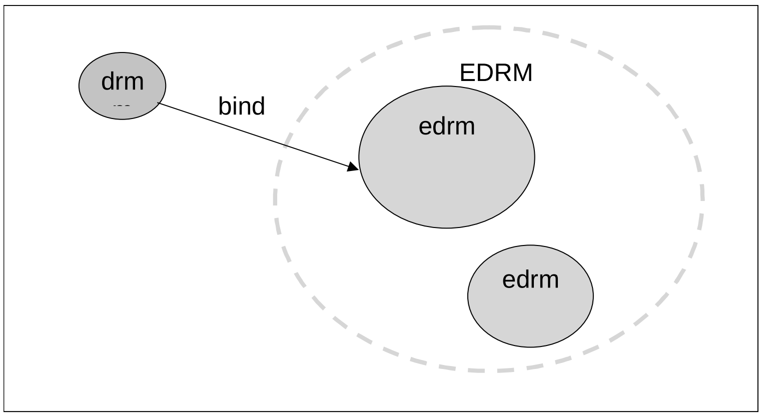

Figure 6 - Data Resource Manager

Relationships:

- A drm is associated with a single edrm.
- An edrm is associated with zero or more drm.

3.1.6 Data Resource

In the data services world there are *Data Resources* (DR). Each dr (dr $\in$ DR) is a grid service and has the usual properties of a grid service specified by the OGSI spec.

A dr represents an edr in the context of an edrm, and is created and inserted into the set DR by binding it to an edr, as illustrated in Figure 7.

A dr exposes the point of contact to an edr, and defines valid operations that may be performed against the edr through the creation of a data access session. A dr also exposes metadata about the structure and content of the edr that it represents.

The dr can act as a notification source for notifications associated with data changes in the data resource.

The lifetime of a dr is managed using the usual OGSI mechanisms. Termination of a dr has no impact on its associated edr. Conversely, if external forces terminate an edr, an associated dr remains extant for its destined life and issues appropriate error responses and notifications.

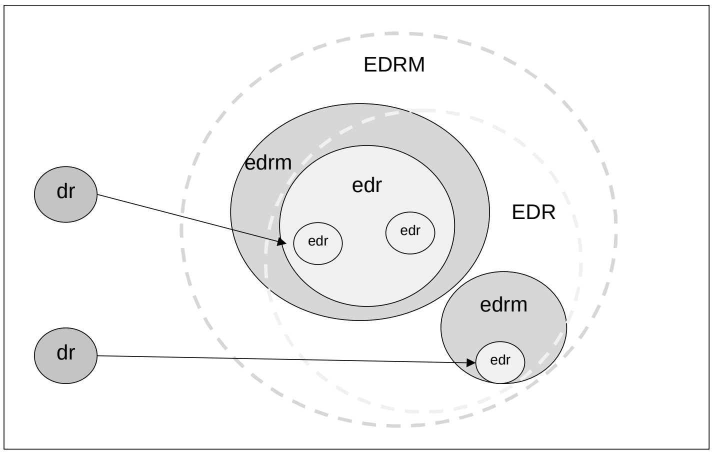

Figure 7 - Data Resource

Relationships:

- A dr is associated with a single edr.
- An edr is associated with zero or many dr.

3.1.7 Data Set

In the data services world there are *Data Sets* (DS). Each ds (ds ∈ DS) is a grid service and has the usual properties of a grid service specified by the OGSI specification.

A ds is an artifact created for the purposes of the DAIS specification and is not restricted to any particular implementation of an eds. The eds may only be modified through interfaces provided by a ds. The GSH of the ds serves to identiy the eds that the ds represents.

A ds has a type based on the type of the eds and exposes data access operations depending on its type.

A ds is introduced into DS either by binding it to eds or by creating a ds and populating it. A ds can be created empty and populated and depopulated using the data access operations.

When a ds is moved/copied (using OGSA mechanisms not yet defined) the eds is moved/copied with it.

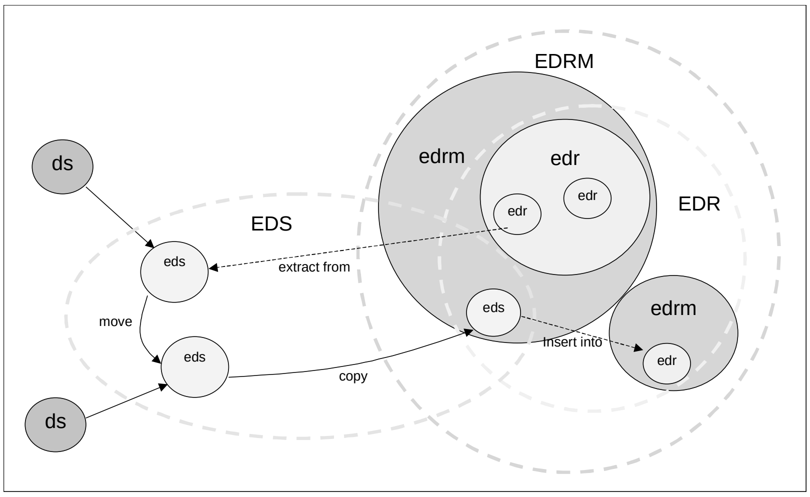

Figure 8 - Data Set

Relationships:

- A ds is associated with a single eds
- An eds is associated with zero or one ds.

3.1.8 Data Activity Session

In the data services world there are *Data Access Sessions* (DAS). Each das (das $\in$ DAS) is a grid service and has the usual properties of a grid service specified by the OGSI spec.

A das is created by, and is closely associated with, a dr through interfaces exposed by the dr.

A das exposes operations that can be performed against the dr that created it. In this way the das defines the mode of operation of data access requests, as illustrated in Figure 9. This model assumes a client server mode of operation, where client requests are satisfied by server actions. Other models could of course replace this, for example, a subscription model could be supported by introducing another type of das.

A das provides the context for operations being performed against the dr. This context is defined by the dr that created the das and through interfaces defined on the das. For example, transactional and security context may be configured on an instance-by-instance basis.

The lifetime of a das can be greater than that of a single operation, but only one operation can be active at any one time. Concurrent operations over a dr are represented using multiple das instances. Many das can be active against a single dr.

A das is the bridge between the managed data in the dr and the logically disconnected data in the ds.

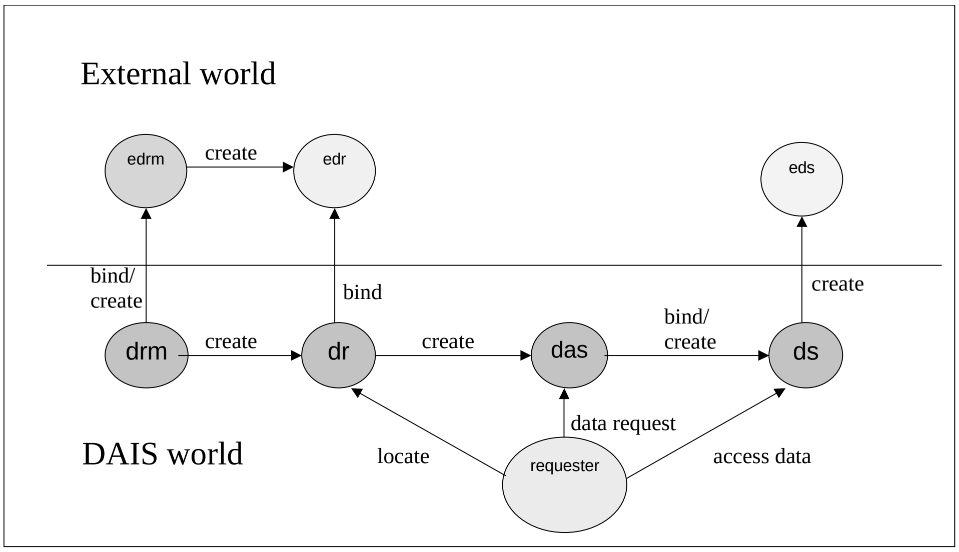

Figure 9 - Data Access Session Associations

Relationships:

- A das is associated with one dr.
- A dr may be associated with zero or more das.
- A das may be associated with zero or more ds.
- A das is associated with only one active dr at any one time.

3.1.9 Data Request

The term data request is introduced to describe the information supplied by a requester to a das. For example, in the case of a das associated with a dr representing a relational database the data request could consist of data manipulation statements, such as SQL queries and updates.

A data request is not represented as a service in its own right. Hence it does not have identity provided by a GSH.

A data request representing a query, when submitted to a das at a particular time, identifies a snapshot of the dr embodied as a ds.

3.2 Mapping to PortTypes and Services

3.2.1 Guiding Principles

The following principles have guided the development of the mapping from conceptual model to portTypes.

OGSI compliant	DAIS presents OGSI compliant mechanisms for accessing data. To that end Version 1 of the OGSI specification is adopted in letter and in spirit.
Extensible and pluggable	The solution presented can be evolved to support new and different data storage systems and access mechanisms.
Easy to understand and apply	For developers the solution exposes simple and regular design patterns. For the end user the emphasis is on consistent behavior with no surprises. The solution generally adopts and supports existing standards and tooling where possible.
Applicable to Grid Services and Web Services	Where practical, the solution supports grid services and web services, where the functionality offered in the grid service setting is an extension of that offered using standard web services.
Access and integration	The solution considers how data may be accessed and integrated both using the mechanisms offered by the physical data resource managers and through DAIS mechanisms.
Implementable	Implementation of the solution is practical with the current state of web and grid services technology and physical data resource managers.
Integrateable into customer scenarios	The solution presents an OGSI compliant solution and will interact with other OGSI services built using this and other OGSI based specifications.
Technology independent	The solution is not dependent on any particular implementation of the OGSI specification. The solution is not dependent on any particular physical data resource manager. DAIS concentrates on those aspects of physical data resources that can be considered to be “standard” and supports extensions that accommodate proprietary extensions.

3.2.2 PortType Extension And Combination

In the spirit of the OGSI specification, this specification defines portTypes. It does not mandate how these portTypes should be combined to form service interfaces.

The OGSI specification describes portType extension as the mechanism that combines portTypes to define a service interface. Using this approach, a service implements a portType that in turn extends from the group of portTypes that describe the function that the service will implement. The DAIS specification also adopts portType extension as its the main extensibility mechanism. It is used extensively to compose portTypes relevant to the different drm, dr, ds and das.

For a given group of portTypes, for example, drm, dr, das or ds, the DAIS specification presents canonical portTypes that either provide a general interface or are specific to particular types of data access. These portTypes can be extended and specialized further. They can also be composed with portTypes defined by other standards, for example, OGSI, or with portTypes defined by applications making use of the DAIS specification.

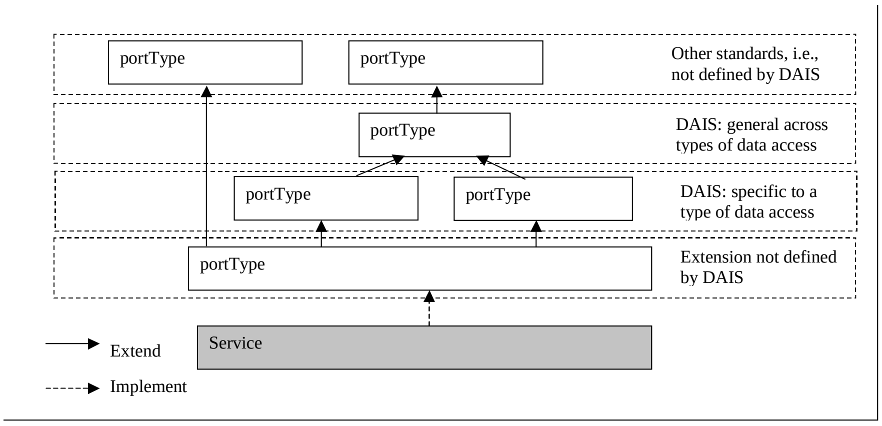
portType
portType
Other standards, i.e., not defined by DAIS
portType
DAIS: general across types of data access
portType
portType
DAIS: specific to a type of data access
portType
Extension not defined by DAIS
Service
Extend
Implement

Figure 10 - PortType Extension And Combination

The specification first considers the general portTypes (Sections 4-7) and then considers extensions which introduce operations and meta-data specific to types of data access (Section 8).

3.2.3 PortType Grouping

The conceptual model presents logical resources that are notionally considered to be OGSI compliant services. These logical resources serve to group portTypes.

Logical Resource	Function
drm	Management of the edrm.
dr	Expose managed data collections.
das	Manipulation operations performed in a session context.
ds	Identify and provide simple access to a collection of data.

The instantiation of four services in order to perform simple data access operations could appear to be a heavyweight solution. To avoid this, appropriate portTypes from these groups can be combined to support simple data access operations on a single grid or web service as required.

An overview of the structure of each of the groups of portTypes is given in the sections that follow.

3.2.4 PortType Mapping Patterns

There are common patterns employed across the groups of portTypes that the logical resources define.

A service hierarchy is formed when the logical resources presented by the conceptual model are considered as services. For example, a drm can create a dr, a dr can create a das, etc. The approach to managing this hierarchy is general.

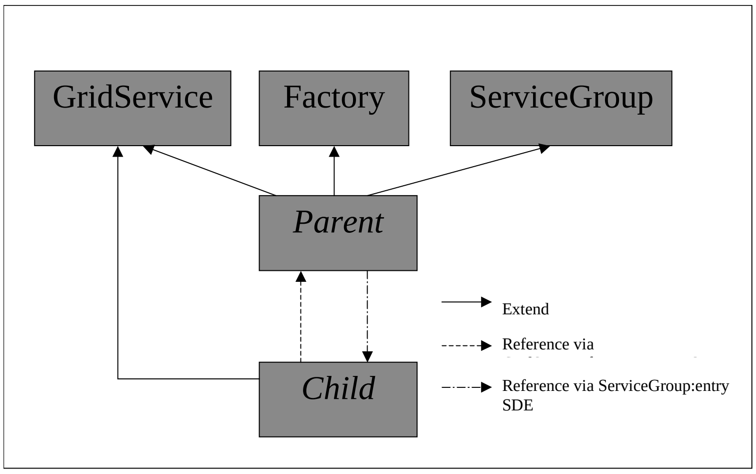

Figure 11 - Parent Child Relationships

The parent service extends Factory and creates child services based on the extensible parameter to the *Factory:createService()* operation. The child maintains a reference back to its parent in the *GridService:factoryLocator* service data element. The parent MAY maintain references to its children in the *ServiceGroup:entry* service data element.

Service data elements characterizing a data service capability are assumed to cascade down to its children. As such, a child service exposes a subset of service data elements to describe the capabilities it supports. Therefore, these capability service data elements are not duplicated in the child service definition. For example, a drm may manage edrs that support certain transactional capabilities. However, a das may choose to support only a subset of these transactional capabilities (or none at all). Hence it publishes an appropriate subset of the SDEs.

There are also patterns employed in order to promote specific operations to general portTypes. For example, generic handling of meta-data is supported using dynamic service data elements.

Generic control of specific edrm and edr is achieved using the same approach that the *Factory:createService()* operation employs, i.e., the operation takes an extensible XML document as a parameter, the format of which is defined in a service data element.

3.2.5 Data Resource Manager

An edrm exists outside the OGSI compliant grid and is affected both by operations from within the grid and by operations external to the grid. The conceptual model suggests a drm service that is constructed to represent the edrm within the grid. This service provides a “point of presence” for the edrm that can be registered in a grid registry and hence located by clients who require access to the edrm.

The DAIS specification is only concerned with the edrm to the extent that it provides the context for an edr and provides a mechanism for obtaining dr. In an OGSI grid, a drm is likely to implement other portTypes that are out of scope of DAIS, for example, management interfaces such as those discussed in the OGSA Plaform document [Foster 03].

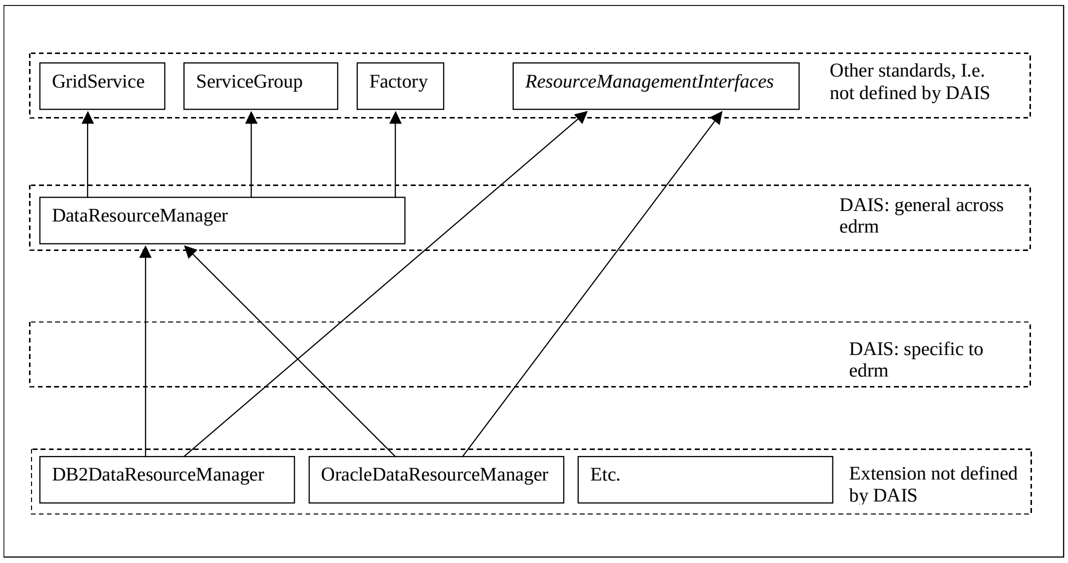
GridService
ServiceGroup
Factory
ResourceManagementInterfaces
Other standards, I.e. not defined by DAIS
DataResourceManager
DAIS: general across edrm
DAIS: specific to edrm
DB2DataResourceManager
OracleDataResourceManager
Etc.
Extension not defined by DAIS

Figure 12 - Data Resource Manager PortType Group

The *DataResourceManager* portType provides the interface for the creation, through extension, of the *Factory* portType, and management, through extension of the *ServiceGroup* portType, of dr instances. It also exposes generic operations for interacting with the edrm in order to set edrm properties and create edr. In defining portTypes for specific edrm that drm services will implement, portTypes defined by other specifications are likely to be included. For example, portTypes associated with the Common Management Model (CMM) and discussed in the OGSA Platform Document [Foster 03]. These extensions are outside the scope of the DAIS spec.

3.2.6 Data Resource

An edr exists outside the OGSI compliant grid and is affected both by operations from within the OGSI grid and by operations external to the OGSI grid. An edr is the point of contact exposed by an edrm. The dr exposes the edr in the OGSI compliant grid.

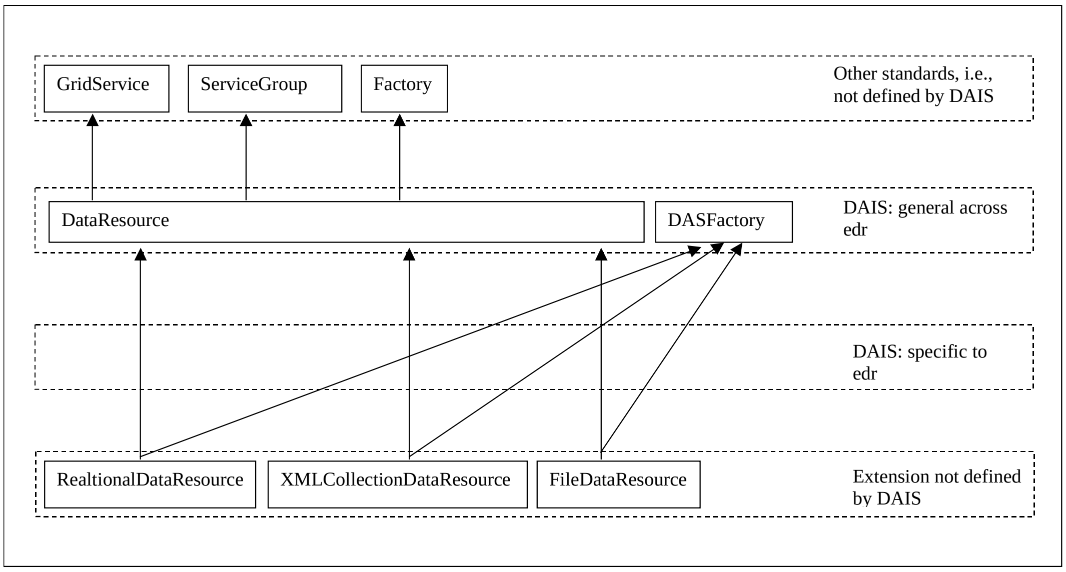

Figure 13 - Data Resource PortType Group

The *DataResource* portType provides the interface for the creation, through extension of the *Factory* portType, and management, through extension of the *ServiceGroup* portType, of das instances. It also exposes a generic mechanism for obtaining information about the structure of the edr it represents.

The *DataResource* portType does not expose any operations that support data access. The dr is used to construct das services that in turn support query operations.

There may be specific types of the dr interface that are required in the future in order to support proprietary features. These will extend *DataResource*. These are outside of the scope of the current DAIS specification.

3.2.7 Data Activity Session

The das allows clients to perform data requests against a single dr. The term “data request” was introduced in the conceptual model to describe the queries or modifications to be performed against a dr.

A das provides the context for the operation against a data request. A das manages such issues as transactions and security. Within this context, a das can only process one data request at a time. Many das instances allow for concurrent data requests.

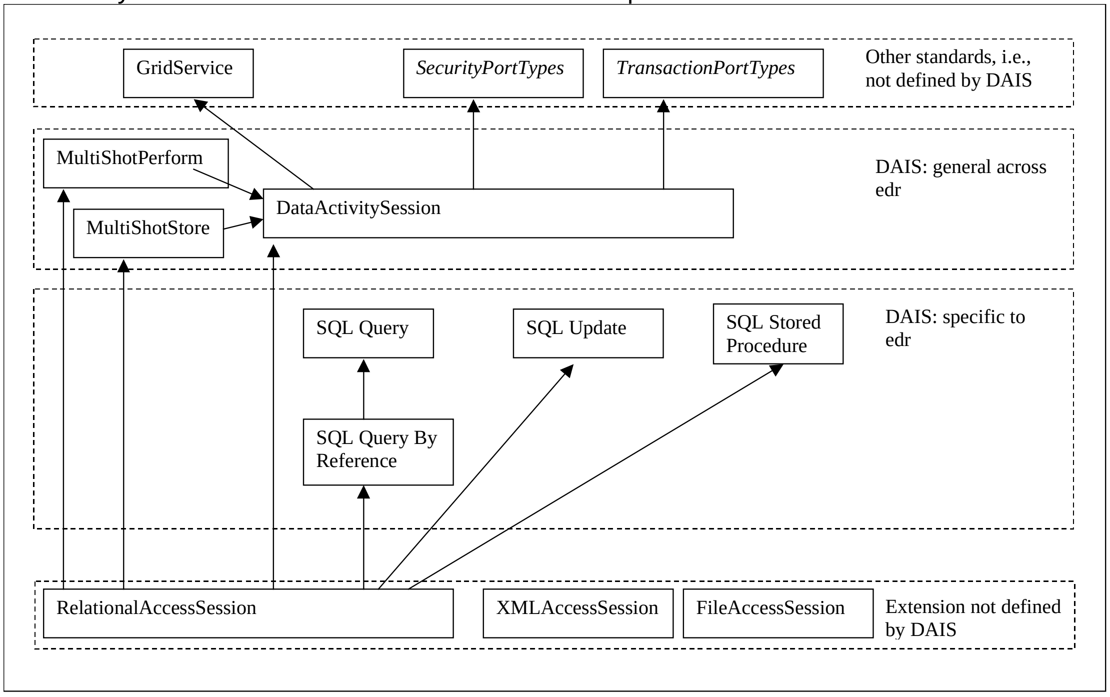

Figure 14 - Data Activity Session PortType Group

The das exposes a number of modes of operation to the client. These modes of operation naturally overlap, and are realized by the portTypes and operations illustrated in Figure 14 for the case of relational databases. Similar portTypes are defined for XML databases in Section 8.3.

1. Pass by value and pass by reference.

An operation may consume or produce data values or references to ds. For example, an operation could return the result of the operation to the client as data values in the response to the request. Alternatively, an operation could return a handle to a ds that holds the data values in the response. The client is then able to use this handle to dictate what happens to the results, for example, the client could pass the service handle to another operation for it to use as its data input.

Whether an operation is pass by value or by reference is indicated by the operation's signature.

2. Grid Services and Web Services.

Current web services technology does not support the concept of a stateful service. Those data access operations defined with a pass-by-value mode of operation are still valid in the web services paradigm. This is because, unlike pass by reference, they do not rely on references to services that contain the data being consumed or produced.

The operations defined against a das are split into portTypes that contain only pass-by-value operations and "extended mode" portTypes that contain pass-by-reference operations. The pass-by-value portTypes can be used without the grid service extensions defined by the OGSI specification.

3. Synchronous and asynchronous.

A synchronous operation performs the operation, populates a ds as required, and then returns the ds handle or data value to the client. An asynchronous operation creates a ds, returns the handle to the client, and then performs the operation that populates the ds. An asynchronous operation is used in the context of a data request where it may be desirable to hide the latency of an operation so that, in effect, the ds is populated in the background while other non-related operations are performed. At some later point, the ds produced can be consumed. This means that the operation that intends to consume the ds is blocked until the ds is ready to be consumed. However, as it may be possible to start consuming data from a ds before it is fully populated, this allows the ds to deliver data incrementally.

In the synchronous case the client is aware that they can send further requests to the das right away. By contrast, in the asynchronous case, the client must query the das to find out when the das is free for further requests.

Whether an operation that supports pass-by-reference is synchronous or asynchronous is indicated by the operation signature. An operation that is pass-by-value is, by its nature, synchronous.

4. Single-shot and multi-shot.

Submitting a query to a das and retrieving the data as a result is an example of a single-shot operation. A multi-shot mode of operation is also supported, which allows multiple activities to be combined into a single data request. Hence, for example, a query activity can be combined with transformation and delivery activities. The specification describes one possible approach to the coordinated execution of a collection of operations by a service.

The operations that manage the submission and management of multi-shot data requests are collected together in the *MultiShotStore* and *MultiShotPeform* portTypes. All other operations defined in this specification are single-shot.

3.2.8 Data Set

An eds is an artifact of data requests within the OGSI compliant grid and is unaffected by operations external to the OGSI grid. A ds is a grid identity for an eds.

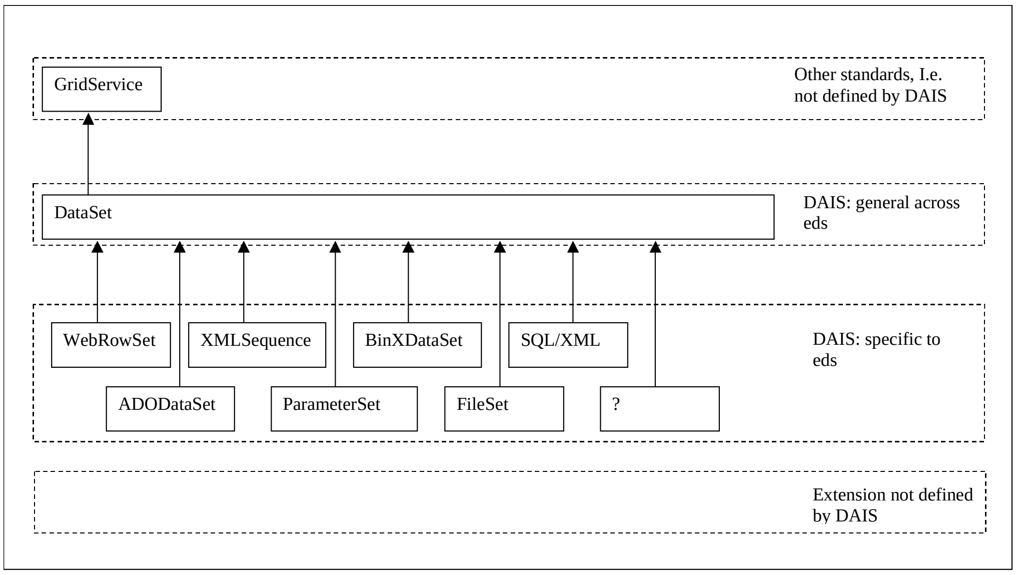

Figure 15 - Data Set PortType Group

The majority of operations associated with a *DataSet* are specific to the type of data that the ds represents. These operations are captured in appropriate extended portTypes.

3.3 Related Concepts

3.3.1 Metadata

The specification includes references to metadata expressed as port types that incorporate Service Data Elements (SDEs), which in turn use XML schema definitions. The specification emphasizes general-purpose metadata for describing and accessing data and metadata in flexible ways. Grid systems do require application specific metadata, even at the data instance level, e.g., to retrieve detailed data instance provenance information. Application-specific metadata will be described in later versions of this specification or in a separate specification.

The metadata topics of interest in this specification can be classified into metadata to support the following constructs:

- Data Resource Manager, e.g., capabilities of drm.
- Data Resource, e.g., structural description of a dr.
- Data Activity Session, e.g., SQLRequest that can be used in a das.
- Data Set, e.g., JDBCResultSet format returned from a client.

Some of the general-purpose standards required by the specification appear in other standards documents. For example:

- Data Activity Session, e.g., XQueryRequest used in a das defined in W3C XQueryX.
- Data Set, e.g., WebRowSet format returned to a client, defined in JCP JSR 114.

3.3.2 Notification

Grid applications will generally rely on notifications to support the loosely coupled soft state nature of their construction. This is also true of data grids. DAIS applications will rely on notification of a number of items, including:

- Data creation, update and deletion.
- Schema changes.
- Request progress and completion.
- Service creation and deletion.

Some notifications will rely on the SDE based notification mechanisms outlined in the OGSI specification. For example, the status of requests submitted to the das.

Some notifications that DAIS must consider are not well supported by this mechanism. For example, data changes are captured in relational databases using triggers, and while it is possible to create a "dataChanged" or "trigger" SDE against which a notification can be raised the implications of this are not well understood.

Further investigation is required to define the full range of notification support.

3.3.3 Security

There are many aspects to security for data access in a grid, e.g., database access security, service access security. In general, this specification will adopt the mechanisms specified by the GGF OGSA Security Working Group. There are areas of security that DAIS or a related data specification will specify, e.g., for linking OGSA security with underlying database security and for mapping database users to grid users.

3.3.4 Transactions

Grid applications will have access to language or requests, such as those described in WS-Transaction [Cabrera 02-b], to specify: transaction start, transaction end (commit, rollback) and intermediate checkpoints (points to rollback to). Resource managers and Coordinators will have access to language or requests, such as those described in WS-Transaction and WS-Coordination [Cabrera 02-a] to indicate participation in a transaction, disposition of their part of the transaction. e.g., committed, rolled back, in doubt, etc. In a transactional environment, when a grid application accesses multiple recoverable resources that support the transaction concept, all resources updated by the grid application within a single transaction are committed or rolled back. Coordination context tokens flow between the calling application, the resource managers being coordinated and the coordinating system, in accordance with WS-Coordination.

Within the context of this specification, all recoverable dr that can be coordinated in transactions are managed by recoverable drm such as database managers. This specification does not introduce additional recoverable resources outside the domain of the resource managers it describes how to access through grid services. If a ds produced from a data request is recoverable and able to be part of a transaction initiated by a grid application, it will be considered to be within the domain of a recoverable drm.

The coordination of transactions across grid and non-grid resources is not considered.

There are a variety of possible transaction options for grid data access. For example:

- Each data request executes in a separate transaction.
- Multiple data requests to a single das execute in a single transaction, but requests across multiple das are not coordinated. Here the das must expose transaction control requests, but a separate coordinator system is not required.
- Multiple requests to multiple das are coordinated. To support this scenario, a single transaction context must be transmitted through multiple das.

Further investigation is required to define the full range of transaction support and its impact on the conceptual model.

4. Data Resource Manager

A *Data Resource Manager* (drm) represents a running External Resource Manager for the purposes of data access and integration. A drm handle may be able to be located in a registry when discovering External Data Resources.

A *DataResourceManager* extends the following portTypes:

- *GridService* portType.

- *DataResourceManager* portType.

DataResourceManager, in turn, forms a base portType that can be extended to represent specific types of edrm, for example, DB2 or Oracle 9i, as illustrated in Figure 12.

4.1 DataResourceManager: Service Data Declarations

The *DataResourceManager* portType includes the following Service Data Element types.

- *externalDataResourceManager***:** a description of the edrm this drm represents; includes product name, version and vendor name.

```
<sd:serviceData name=”externalDataResourceManager”
                type=”dais:ExternalDataResourceManagerType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”>
```

- *physicalProperties:* physical constructs; includes space used, buffers.

```
<sd:serviceData name=”physicalProperties”
                type=”dais:PhysicalPropertiesType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *softwareCapabilities:* capabilities of the software; includes software version, service packs, patch level.

```
<sd:serviceData name=”softwareCapabilities”
                type=”dais:SoftwareCapabilitiesType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

- *featuresInstalled:* description of installed features; includes product enhancements, features, extra functionalities.

```
<sd:serviceData name=”featuresInstalled”
                type=”dais:FeaturesInstalledType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *featuresActivated:* lists the items from *featuresInstalled* that are activated in the installation.

```
<sd:serviceData name=”featuresActivated”
                type=”dais:FeaturesActivatedType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *ownership***:** Signed certificate.

```
<sd:serviceData name=”Ownership”
```

```
                type=”dais:OwnershipType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *securityCapabilities:* lists the security capabilities are provided by the resource manager..

```
<sd:serviceData name=”securityCapabilities”
                type=”dais:SecurityCapabilitiesType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *transactionCapabilities:* lists the transactional capabilities are provided by the resource manager.

```
<sd:serviceData name=”transactionCapabilities”
                type=”dais:TransactionCapabilitiesType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *externalDataResource*: the names of available EDRs.

```
<sd:serviceData name=”externalDataResource”
                type=”dais:ExternalDataResourceType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

4.2 DataResourceManager: Operations

No operations are currently defined on *DataResourceManager*.

5. Data Resource

A *DataResource* represents an external data resource (edr). The dr will exhibit a type based on the type of the edr. *DataResource* forms the base object, which may be extended to represent specific types of edrs, e.g. RelationalDataResource or XMLDataResource.

A dr extends the following portTypes:

- *GridService* port Type.
- *DataResource* portType.
- *DASFactory* portType.

The *ogsi:factoryLocator* SDE (inherited from the GridService portType) may refer to the DataResourceManager that created this dr.

5.1 DataResource: Service Data Declarations

The following service data declarations are cascaded down from the drm that manages this dr, as appropriate, as discussed in Section 3.2.4.

- SoftwareProperties.

- FeaturesInstalled.
- FeaturesActivated.
- SecurityCapabilities.
- TransactionCapabilities.

Additionally, the DataResource portType includes the following serviceData elements:

- *daisObjectName*: Possible object types. The QNames for each object type definition supported by this service instance, such as *databaseSchema*, *storedProcedure*, *collectionStructure*, *trigger* as defined in Sections 8.2.2.1 and 8.3.2.1.

```
<sd:serviceData name=”daisObjectName”
                type=”xsd:QName”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”constant”
                modifiable=”false”
                nillable=”false”/>
```

- *object definitions:* Describes an object. This is a dynamic SDE that is specific to the service instance. The object name MUST be referenced by a *daisObjectName* SDE.

5.2 DataResource: Operations

The *DataResource* portType defines no operations.

5.3 DASFactory port type

The *DASFactory* portType extends the OGSI *Factory* portType. The *createServiceExtensibility* SDE inherited from the *Factory* portType must conform to the XML schema type *DASCreateDocumentType*.

In addition, the *DASFactory* portType includes the following serviceData elements:

- *dataActivitySessionType*: Description of available DAS types. This includes information on supported operations, drivers and language capabilities.

```
<sd:serviceData name=”dataActivitySessionType”
                type=”dais:DataActivitySessionType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

6. Data Activity Session

A *Data Activity Session* (das) provides the context for data request operations performed against a *Data Resource*. A das exposes metadata about the activities it supports. Only one operation may be run at a time. A das is created by, and is closely associated with, a data resource. A das may create or read data from *DataSets*.

A DAS can be associated with exactly one DR.

The DAS extends the following port types:
- o *GridService* portType.

The DAS may also extend additional portTypes for security and transaction management.

The *ogsi:factoryLocator* SDE (inherited from the *GridService* portType) must refer to the *DataResource* that created this *DataActivitySession*.

6.1 DataActivitySession: Service Data Declarations

The SDEs describing *object names* and *definitions* are cascaded down from the dr that created this das, as appropriate, as discussed in Section 3.2.4.

The DAS portType includes the following serviceData elements.

- *status*: status information on the running request.

```
<sd:serviceData name=”status”
                type=”dais:StatusType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

- *languageCapabilities*: language capabilities.

```
<sd:serviceData name=”languageCapabilities”
                type=”dais:LanguageCapabilitiesType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

- *driver*: the driver used to connect to the data resource.

```
<sd:serviceData name=”driver”
                type=”dais:DriverType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

6.2 DataActivitySession: Operations

No operations are currently defined on *DataActivitySession*, although extensions of *DataActivitySession*, as illustrated in Figure 14, can be expected to define operations for submitting requests.

6.3 Multiple Activity PortTypes

This section describes the *MultiShotPerform* and *MultiShotStore* portTypes, which support the coordinated execution of a collection of activities.

6.3.1 MultiShotPerform

6.3.1.1 Service Data Declarations

This subsection describes service data that is specific to the coordinated execution of a collection of activities.

- *dataRequestSchema*: the schema of the data request document.

```
<sd:serviceData name=”dataRequestSchema”
                type=”xsd:SchemaType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”false”/>
```

- *activityType*: activity types supported by this DAS.

```
<sd:serviceData name=”activityType”
                type=”dais:ActivityTypeType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

6.3.1.2 Operations

This subsection describes operations that are specific to the coordinated execution of a collection of activities.

MultiShotPerform::perform
Perform a request on a DataActivitySession.
Input:
- *dataRequest*: The document that describes the operation to be performed.

Output:
- *dataRequestResponse*: The document that provides the result.

Fault:
- *SystemException*: An exception report that the client can take no action to resolve
- *UserException*: An exception report that the client can resolve
- *NameNotUniqueException*: All top-level elements in the *dataRequest* document are named. If a name is used that is not unique, in the context of the DAS, an exception is raised.

The behavior of the DAS, following a call to *DataAccessSession::perform*, is characterized by the contents of the *dataRequest* document passed in as a parameter.

A *dataRequest* is of type *DataRequestType*. A *dataRequest* is a collection of linked *activities*, in which *data flow links* can be formed between the outputs of some activities and the inputs of others. In essence, an *activity* represents the invocation of an operation. It has two attributes, a *name* (which is local to the activity, and which must be unique within the activity) and an *operation* (the name of a WSDL operation). An activity also has an optional *input* element that consists of a collection of *part* elements. Each *part* element provides a value for one of the parts of the message that is the input to the operation in the WSDL. An activity need not give values for all of the parts of the input message. The other parts can be given values using links, as described below.

A *link* is an element that associates one of the parts of the output of one activity with one of the parts of the input of another. The presence of a link means that data will flow from the output of one activity to form the input of another. For example, the following *dataRequest* contains two activities that describe invocations of *sqlQuery* and *deliverByFTP* operations. The WSDL for the *sqlQuery* has a single part to its input message, called *expression*, which is instantiated in the activity with a query string. The *deliverByFTP* operation has two parts to its input message, namely *theDestination* and *theData*. The value for *theDestination* is provided as part of the activity, whereas the value for *theData* is obtained from the *result* part of the output of the evaluation of the *sqlQuery.* A link is also used to connect the parts of the parameters provided when executing a request to the activities in the request that consume the parameters. The parameters of a request are of type *DataRequestParameterType*.

```
<dataRequest name=”myRequest”>
  <activity name="theQuery" operation="sqlQuery">
```

```
    <input>
      <part partName="query"> select * from tab </part>
    </input>
  </activity>

  <activity name="theDeliver" operation="gftpDelivery">
    <inputs>
      <part partName="theDestination"> ftp://ftp.man.ac.uk/myfile </part>
    </inputs>
  </activity>

  <link src="theQuery" scrPart="WebRowSet"
        dest="theDeliver" destPart="value"/>
</dataRequest>
```

The invocation of a perform operation has the following characteristics:

1. A das evaluates a single request at a time. Thus a *SystemException* fault is generated if the das is evaluating an earlier request when a perform request is received.
2. A *dataRequest* evaluates within a single transaction over its associated data resource, and constitutes the complete behaviour of that transaction.
3. A *dataRequest* consists of a connected, directed acyclic graph, with a unique sink node that is reachable from all other vertices (such a sink node is sometimes referred to as a supersink - http://www.nist.gov/dads/). Where there is branching in the graph, parallel branches are evaluated in the order in which their start nodes appear in the *dataRequest*.
4. The result of a *dataRequest* that evaluates successfully is of type DataResponseType, which contains the result of the evaluation of the unique sink node. The perform operation returns a result to its requester when the operation invoked by its unique sink node returns a result. As such, a *dataRequest* inherits the property of being synchronous or asynchronous from its unique sink node. The *timeStamp* attribute of the *dataResponse* is the time when the associated *dataRequest* was received.

MultiShotPerform::terminate
Terminates the currently running request.

Input
- *None* (an empty input message)

Output
- *Report*

Fault(s)
- *NoRunningRequestFault*
- *Fault*: any other fault

6.3.2 MultiShotStore

6.3.2.1 Service Data Declarations

This subsection describes service data that is specific to the storage of documents supporting coordinated execution of a collection of activities.

- *storedRequest*: details of the requests stored in the session.

```
<sd:serviceData name=”storedRequest”
                type=”dais:RequestType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

6.3.2.2 Operations

MultiShotStore::storeRequest
Stores a request document for deferred execution.

Input
- *DataRequest*

Output
- *Name:* the name of the stored request document

Fault(s)
- *RequestTypeNotSupportedFault*:
- *Fault*: any other fault.

MultiShotStore::executeRequest
Executes a stored request.

Input
- *RequestName*: the id of the request document that is to be executed
- *Parameters:* parameters referred to by links in the request

Output
- *DataRequestResponse*: results of the data request

Fault(s)
- *Fault*: any other fault

MultiShotStore::removeRequest
Removes a stored request.

Input
- *RequestName*: the name of the request document that is to be removed

Output
- *None* (an empty output message, acknowledging that the request has been removed)

Fault(s)
- *NoSuchRequestFault*:
- *Fault*: any other fault

7. Data Set

A *DataSet* (ds) represents an external data set (eds). The ds exhibits a type based on the type of the eds. DataSet forms the base object which may be extended to represent specific types of external data sets, e.g., the results of a SQL query or an XML query; these represent a result set which results from running a query against a data resource.

A DS extends the following portTypes:
- *GridService* portType
- *DataSet* portType

The *ogsi:factoryLocator* SDE (inherited from the *GridService* portType) may refer to the DataActivitySession that created this dataset.

7.1 DataSet: Service Data Declarations

The *DataSet* portType includes the following serviceData elements.
- *structuralCharacteristics*: type and format.

```
<sd:serviceData name=”structuralCharacteristics”
                type=”dais:StructuralCharacteristicsType”
                minOccurs=”0” maxOccurs=”1”
                mutability=”static”
                modifiable=”false”
                nillable=”false”/>
```

- *size*: size information.

```
<sd:serviceData name=”size”
                type=”dais:SizeType”
                minOccurs=”0” maxOccurs=”1”
                mutability=”static”
                modifiable=”false”
                nillable=”false”/>
```

- *readOnce*: boolean indicating whether this *DataSet* can be read once or multiple times.

```
<sd:serviceData name=”readOnce”
                type=”xsd:boolean”
                minOccurs=”1” maxOccurs=”1”
                mutability=”static”
                modifiable=”false”
                nillable=”false”/>
```

7.2 DataSet: Operations

7.2.1 DataSet::get

Returns all the data managed by a DataSet to a client.

Input
- *None* (an empty input message)

Output
- *Result*: any XML data

Fault(s)
- *Fault*: any other fault

DataSets will have move and copy operations, e.g., to enable a DataSet service to execute on a different system. DataSet operations will be implemented through OGSA agreed mechanisms for moving and copying services (yet to be defined).

7.3 DataSet Extensions

The following portTypes extend the DataSet portType.
- WebRowSet (http://java.sun.com/xml/ns/jdbc/webrowset.xsd).
- SQLXMLRowSet (http://standards.iso.org/iso/9075/2002/12/sqlxml.xsd).
- ADODataSet.
- XPathResultSet.
- XQueryResultSet.
- ParameterSet.
- XMLSequence.
- BinXDataSet.
- FileSet.

8. Data Service Realizations

8.1 Overview

This section describes how the conceptual model from Section 3, and subsequently elaborated in Sections 4 to 7 as a collection of portType and service data declarations, can be realized by associating the generic model with specific functionalities, such as those provided by a particular *data resource manager*, a *data resource* bound to a specific type of *external data resource,* and the data model specific operations that a *data activity session* can support. Subsections are dedicated to each data model, in this instance relational and XML, and to additional concrete capabilities such as data delivery and transformation.

8.2 Relational Database Access

8.2.1 Data Resource Manager

8.2.1.1 Service Data Declarations

This subsection describes service data that is specific to a *data resource manager* that is bound to a relational based *external data resource*.

No such service data declarations are currently defined.

8.2.1.2 Operations

This subsection describes operations that are specific to a *data resource manager* that is bound to a relational based *external data resource manager.*

No such operations are currently defined.

8.2.2 Data Resource

8.2.2.1 Service Data Declarations

This subsection describes service data that a *DataResource* bound to an *external data resource* employing a relational model SHOULD support.

- *databaseSchema***:** Describes the database schema of the relational database, such as tables, columns, column types, keys, etc.

```
<sd:serviceData name=”databaseSchema”
                type=”DatabaseSchemaType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *storedProcedure***:** Describes the name, input and output types of the stored procedures made available by the data resource.

```
<sd:serviceData name=”storedProcedure”
                type=”StoredProcedureType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *trigger***:** Describes the name and definition of the triggers deployed in the data resource.

```
<sd:serviceData name=”trigger”
                type=”TriggerType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *index***:** Describes the names and definitions of the indexes deployed in the data resource.

```
<sd:serviceData name=”indexes”
                type=”IndexType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
```

- *languageCapabilitie***:** Describes the dialect of SQL supported by this data resource.

```
<sd:serviceData name=”languageCapabilities”
                type=”LanguageCapabilitiesType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
```

- *userDefinedType*: Describes the names and definitions of the user defined types deployed in the data resource.

```
<sd:serviceData name=”userDefinedType”
                type=”UserDefinedTypesType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
```

8.2.2.2 Operations

This subsection describes operations that are specific to a *data resource* that is bound to a relational based *external data resource.*

No such operations are currently defined.

8.2.3 Data Activity Session

This subsection describes the operations that are specific to a *RelationalAccessSession* that interacts with a relational based *external data resource*. Three categories of operation have been identified based on the way in which results are processed:

- Data is returned by value directly as a response to the operation. This is the default mode of operating for all operations.
- Data is returned indirectly, that is a service handle to a *DataSet* is returned, with which the client can subsequently interact. In this instance the operation name is suffixed with the identifier "*ByRef*". Two further modes of operation are distinguished:

    - The reference to the *DataSet* is returned *before* the *DataSet* has been populated with the required data. In this case the operation is said to be operating in an *asynchronous* manner. Appending the identifier "Async" to the operation name denotes this mode of operation.
    - The reference to the *DataSet* is returned *after* the *DataSet* has been populated with the required data. In this instance we say that the operation is operating *synchronously.* Appending the identifier "*Sync*" to the operation name denotes this mode of operation.

The convention is summarized in the table below.

Suffix	Meaning
<operation> (No Suffix)	Operation returns data by value.
<operation>ByRefSync	Operation returns the GSH of a *DataSet* that has been populated with the data resulting from the operation.
<operation>ByRefAsync	Operation returns the GSH of a *DataSet* that has not been populated with data resulting from the operation.

These operations are divided across several portTypes, as illustrated in the following table:

portType	operations
relationalManagement	dropDatabase
	createDatabase
sqlQuery	sqlQuery
sqlQueryByRef	sqlQueryByRefSync
	sqlQueryByRefAsync
sqlUpdate	sqlUpdate
sqlStoredProcedure	sqlStoredProcedure

These base portTypes may be aggregated in different ways to form the service interface

8.2.3.1 Relational Management

8.2.3.1.1 Service Data Declarations

This subsection describes service data that is specific to a *data resource manager* that is bound to a relational based *external data resource*.

No such service data declarations are currently defined.

8.2.3.1.2 Operations

relationalManagement::dropDatabase
Allows a database to be dropped from a relational based data resource.

Input
- *databaseName*: the name of the database that is to be removed.

Output
- *resultStatus*: the status of the removal of the database from the RDBMS if any.

Fault(s)
- *PermissionDenied*: the client does not have sufficient permission to drop the database.
- *NoSuchDatabase*: the database name does not exist.
- *Fault*: any other fault.

relationalManagement::createDatabase
Allows a database to be created within a relational based data resource.

Input
- *databaseName*: the name of the database to be created.
- *databaseSchema* (optional): the database schema or the DDL required to create the tables associated with this database.

Output
- *resultStatus*: output status from the database creation operation if any.

Fault(s)
- *PermissionDenied*: the client does not have sufficient permissions to create the database.

- *Fault*: any other fault

8.2.3.2 SQL Query

8.2.3.2.1 Service Data Declarations

This subsection describes service data that is specific to SQL Queries.

No such service data declarations are currently defined.

8.2.3.2.2 Operations

Direct an SQL Query to a relational based data resource where the result of the query is returned as a WebRowSet value.

sqlQuery::sqlQuery

Input
- *expression*: the SQL query string that is to be run on the data resource.

Output
- *WebRowSet*: output the results in the web row set format.

Fault(s)
- *InvalidQuery*: the supplied SQL is syntactically incorrect or fails during evaluation.
- *Fault*: any other fault.

8.2.3.3 SQL Query By Reference

8.2.3.3.1 Service Data Declarations

This subsection describes service data that is specific to SQL query requests that return the GSH of a *DataSet*.

No such service data declarations are currently defined.

8.2.3.3.2 Operations

Direct an SQL Query to a relational based data resource where the result of the query is returned as the GSH of a *DataSet*.

sqlQueryByRef::sqlQueryByRefSync/sqlQueryByRef::sqlQueryByRefAsync

Direct an SQL Query to a relational based data resource and return a DataSet handle.

Input
- *expression*: the SQL query string that is to be run on the data resource.

Output
- *DataSetHandle*: returns the GSH of a *DataSet* which, in the synchronous case will be populated with the data returned by the SQL query, while in the asynchronous case, the *DataSet* is not populated with the data resulting from the query before the operation exists.

Fault(s)
- *InvalidQuery*: the supplied SQL is syntactically incorrect or has some other problem an SQL update statement has been supplied to the query interface.
- *Fault*: any other fault.

8.2.3.4 SQL Update

8.2.3.4.1 Service Data Declarations

This subsection describes service data that is specific to SQL Updates.

No such service data declarations are currently defined.

8.2.3.4.2 Operations

sqlUpdate::sqlUpdate
Direct an SQL update to a relational based data resource.

Input
- *expression*: the SQL update string.

Output
- *updateStatus*: the result of the update operation.

Fault(s)
- *InvalidQuery*: the SQL supplied to do the update is incorrect or an SQL query statement is being supplied to this operation.
- *Fault*: any other fault.

8.2.3.5 SQL Stored Procedure

8.2.3.5.1 Service Data Declarations

This subsection describes service data that is specific to SQL Stored Procedures.

The SDEs describing storedProcedures are cascaded down from the parent dr as appropriate.

8.2.3.5.2 Operations

sqlStoredProcedure::sqlStoredProcedure
Invoke a stored procedure in a relational based data resource.

Input
- *procedureName*: the name of the stored procedure that is to be used.
- *parameterList*: a list of parameters to be used with the stored procedure.

Output
- *relationalResults*: output of the stored procedure.

Fault(s)
- *NoSuchProcedureName*: the *procedureName* has not been recognized.
- *PermissionDenied*: insufficient authority to invoke the procedure.
- *Fault*: any other fault.

8.3 XML Database Access

8.3.1 Data Resource Manager

8.3.1.1 Service Data Declarations

Here we describe any service data that would be specific to a *data resource manager* that is bound to an XML based *external data resource*.

No such service data declarations are currently defined.

8.3.1.2 Operations

Here we describe any operations that would be specific to a *data resource manager* that is bound to an XML based *external data resource*.

No such operations are currently defined.

8.3.2 Data Resource

8.3.2.1 Service Data Declarations

The following describes service data that a *data* resource bound to an *external data resource* employing an XML model SHOULD support.

- *languageCapabilities*: Describes the dialect of XPath, XQuery supported by this dr.

```
<sd:serviceData name=”languageCapabilities”
                type=”LanguageCapabilitiesType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
```

- *collectionStructure*: Describes the sub-collections of an XML database collection.

```
<sd:serviceData name=”collectionStructure”
                type=”CollectionStructureType”
                minOccurs=”1” maxOccurs=”1”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

- *collectionSchema*: XML schemas associated with an XML database collection.

```
<sd:serviceData name=”collectionSchema”
                type=”CollectionSchemaType”
                minOccurs=”0” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
                nillable=”true”/>
```

8.3.2.2 Operations

Here we describe any operations that would be specific to a *data resource* that is bound to an XML based *external data resource*.

No such operations are currently defined.

8.3.3 Data Activity Session

This subsection describes the operations that are specific to a *XMLAccessSession* that interacts with a XML based *external data resource*. These operations are divided across several portTypes, as illustrated in the following table:

portType	operations
collectionManager	createCollection
	removeCollection
resourceManger	createResource
	removeResource
xUpdate	xUpdateStatment
xPath	xPathStatement
xPathByRef	xPathStatementByRefSync
	xPathStatementByRefAsync
xQuery	xQueryStatement
xQueryByRef	xQueryStatementByRefSync
	xQueryStatementByRefAsync

8.3.3.1 Collection Manager

8.3.3.1.1 Service Data Declarations

This subsection describes service data that is specific to XML Collection Management.

No such service data declarations are currently defined.

8.3.3.1.2 Operations

collectionManager::createCollection
Allows a new collection to be created within an XML based data resource.
Input

- *Name*: the name of the collection to be created.
- Schemas (optional): A list of schemas that the documents in the new collection must satisfy.

Output

- *createStatus*: status of the create operation.

Fault(s)

- *PermissionDenied*: do not have privileges to create a collection.
- *Fault*: any other fault.

collectionManager::removeCollection
Remove a collection from an XML based data resource.
Input

- *Name*: the name of the collection to be removed.

Output

- *removeStatus*: status of the remove operation.

Fault(s)

- *NoSuchCollection*: *Name* used for the collection has not been recognized.
- *PermissionDenied*: insufficient privileges to remove the collection.
- *Fault*: any other fault.

resourceManager::createResource
Create resource within a collection.

Input

- *resourceName*: the name of the resource to be created within a collection specified implicitly by the data resource the DAS is associated with or explicitly by adding the collection name in the argument list.
- *resourceData*: the data that is to go in this resource.
- *collectionName* (optional): the name of the collection where the resource is to be created.

Output

- *createStatus*: status of the create resource operation.

Fault(s)

- *InvalidResource*: if an XML Schema is associated with the collection and the resource to be added does not conform.
- *Fault*: any other fault.

resourceManager::removeResource
Remove a resource within a given collection.

Input

- *resourceName*: the name of the resource to be removed from a collection.
- *collectionName* (optional): the name of the collection where the resource is to be removed.

Output

- *removeStatus*: status of the remove resource operation.

Fault(s)

- *NoSuchResource*: the name of the data resource to be removed does not exist.
- *PermissionDenied*: client does not have sufficient permissions to remove this resource.
- *Fault*: any other fault.

8.3.3.2 XML Update

8.3.3.2.1 Service Data Declarations

This subsection describes service data that is specific to XML Updates.

No such service data declarations are currently defined.

8.3.3.2.2 Operations

xUpdate::xUpdateStatement
An XUpdate operation is to be used to update a resource in an XML based data resource.

Input

- *expression*: the XUpdate expression that is to be used to update the resource.
- *collection* (optional): the collection that contains the resource to which the XUpdate is to be applied.
- *resourceId* (optional): the identifier for the resource to which the update is to be performed.

Output

- *updateStatus*: status of the XUpdate operation.

Fault(s)

- *UpdateFailed*: the XUpdate operation failed.
- *NoSuchResource:* resource specified on which the XUpdate expression is to be applied does not exist.
- *Fault*: any other fault.

8.3.3.3 Xpath Queries

8.3.3.3.1 Service Data Declarations

This subsection describes service data that is specific to Xpath queries.

No such service data declarations are currently defined.

8.3.3.3.2 Operations

xPath::xPathStatement

Apply an Xpath query to a resource to an XML based data resource.

Input

- *expression*: the XPath expression that is to be applied.
- *collection* (optional): the collection that contains the resource to which the XPath is to be applied.
- *resourceId* (optional): the identifier for the resource to which the query is to be applied.
- *nameSpaces* (optional): a list associating the namespace prefixes with the actual namespace used in the XPath.

Output

- *resultSet*: the output of the XPath expression.

Fault(s)

- *InvalidQuery:* the XPath expression is syntactically incorrect.
- *NoSuchResource:* resource specified on which the XPath expression is to be applied does not exist.
- *Fault*: any other fault.

8.3.3.4 XPath By Reference Queries

8.3.3.4.1 Service Data Declarations

This subsection describes service data that is specific to XPath queries.

No such service data declarations are currently defined.

8.3.3.4.2 Operations

xPathByRef::xPathStatementByRefSync/xPathStatmentByRefAsync

Apply an XPath query to a resource to an XML based data resource. The result set is returned as the GSH of a DataSet.

Input

- *expression*: the XPath expression that is to be applied.
- *collection* (optional): the collection that contains the resource to which the XPath is to be applied.
- *resourceId* (optional): the identifier for the resource to which the query is to be applied.
- *nameSpaces* (optional): a list associating the namespace prefixes with the actual namespace used in the XPath.

Output

- *DataSetHandle*: returns the GSH of a *DataSet* which, in the synchronous case will be populated with the data returned by the XPath query, while in the asynchronous case, the *DataSet* is not be populated with the data resulting from the query before the operation exists.

Fault(s)

- *InvalidQuery:* the XPath expression is syntactically incorrect.
- *NoSuchResource:* resource specified on which the XPath expression is to be applied does not exist.
- *Fault*: any other fault.

8.3.3.5 XQuery Queries

8.3.3.5.1 Service Data Declarations

This subsection describes service data that is specific to XQuery queries.

No such service data declarations are currently defined.

8.3.3.5.2 Operations

xQuery::xQueryStatement
Perform an XQuery on an XML resource within an XML based data resource. If no collection or list of resources is supplied then the XQuery will be applied to all resources contained in the data resource the data activity session is dealing with.

Input

- *expression*: the XQuery expression that is to be applied.
- *collection* (optional): the collection that contains the resource to which the XQuery is to be applied.
- *resourceIds* (optional): a list of resources over which the XQuery expression is to be applied.
- *nameSpaces* (optional): a list associating the namespace prefixes with the actual namespace used in the XQuery.

Output

- *resultSet*: output of the XQuery expression returned directly to the client.

Fault(s)

- *InvalidQuery*: the XQuery expression is invalid.
- *NoSuchResource:* resource specified on which the XQuery expression is to be applied does not exist.
- *Fault*: any other fault.

8.3.3.6 XQuery By Reference Queries

8.3.3.6.1 Service Data Declarations

This subsection describes service data that is specific to XPath queries.

No such service data declarations are currently defined.

8.3.3.6.2 Operations

xQueryByRef::xQueryStatementByRefSync/xQueryStatementByRefAsync
Perform an XQuery on an XML resource within an XML based data resource. The same conditions apply as in the return by value case above.

Input

- *expression*: the XQuery expression that is to be applied.
- *collection* (optional): the collection that contains the resource to which the XQuery is to be applied.

- *resourceIds* (optional): a list of resources over which the XQuery expression is to be applied.
- *nameSpaces* (optional): a list associating the namespace prefixes with the actual namespace used in the XQuery.

Output

- *DataSetHandle*: returns the GSH of a *DataSet* which, in the synchronous case will be populated with the data returned by the XQuery query, while in the asynchronous case, the *DataSet* is not populated with the data resulting from the query before the operation exists.

Fault(s)

- *InvalidQuery*: the XQuery expression is invalid.
- *NoSuchResource:* resource specified on which the XQuery expression is to be applied does not exist.
- *Fault*: any other fault.

8.4 Data Delivery

8.4.1 Service Data Declarations

The following describes service data associated with data delivery.

- **deliveryStatus:** Describes the progress of active data deliveries.

```
<sd:serviceData name=”deliveryStatus”
                type=”LanguageCapabilitiesType”
                minOccurs=”1” maxOccurs=”unbounded”
                mutability=”mutable”
                modifiable=”false”
```

8.4.2 Operations

The delivery operations described here are only relevant in the multi-shot approach where it may be desirable to deliver data or bring data into a service from an external resource without having to instantiate a dataSet.

portType	operations
urlDelivery	deliverToURL
	deliverFromURL
gftpDelivery	deliverToGFTP
	deliverFromGFTP

urlDelivery::deliverToURL
Delivery of a value to a location identified by a URL.

Input

- *value*: the data that is to be delivered.
- *url:* URL to which the dataSet is to be delivered.

Output

- *deliveryStatus*: status of the delivery operation.

Fault(s)

- *Fault*: any other fault.

urlDelivery::deliverFromURL
Deliver data identified by a URL.

Input
- *url:* URL from which the data is to be obtained.

Output
- *value*: the data.

Fault(s)
- *Fault*: any other fault.

gftpDelivery::deliverToGFTP
Deliver to a host using GridFTP.
Input
- *value:* the data to be delivered.
- *host:* host which is to be delivered to.
- *port:* port to be used by the delivery.
- *file:* file to which delivery is to take place.

Output
- *deliveryStatus*: status of the delivery.

Fault(s)
- *Fault*: any other fault.

gftpDelivery::deliverFromGFTP
Use GridFTP to obtain data.

Input
- *host:* host which data is to be obtained.
- *port:* port to be used to get the data.
- *file:* file to which has the desired data.

Output
- *value*: the data that has been delivered.

Fault(s)
- *Fault:* any other fault.

8.5 Transformation

A single example transformation activity is provided here that could operate with either a relational or XML data model - an XSL-Transformation. Additional portTypes could be created to allow transformations involving DataSets.

8.5.1 XSLTransform::xslTransform

This assumes that some previous operation has produced an XML document that the client wishes to transform to another format.

Input
- *XMLData*: the data that has to be transformed.
- *xsltDocument*: the XSLT document to be employed to be used to effect the transformation.

Output
- *transformedData*: document containing the transformed data.

Fault(s)
- *Fault:* any other fault

9. Conclusion

This document has described a proposal for a collection of Grid data access services, which includes support for multiple database paradigms. The services proposed are Grid services, in that they conform to and make use of the Open Grid Services Infrastructure [Tuecke 03]. The proposal will be discussed and evolved in the context of the DAIS Working Group (http://www.ggf.org/6_DATA/dais_b.htm) of the Global Grid Forum. XML Schema and WSDL definitions for this specification will be made available and incrementally updated on the DAIS Web Site.

Acknowledgement: The DAIS Working Group of the Global Grid Forum is active, and many people have contributed to discussions within the group in recent months, including but not limited to: Bill Allcock, Vijay Dialani, Dieter Gawlick, Shannon Hastings, Stephen Langella, Sastry Malladi, Inderpal Narang, Dave Pearson, Ekkehard Rohwedder, Steve Tuecke and Paul Watson.

Author Information

Mario Antonioletti, EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

Malcolm Atkinson, e-Science Institute, 15 South College Street, Edinburgh EH8 9AA, UK.

Neil P Chue Hong, EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

Amy Krause, EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

Susan Malaika, IBM Corporation, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, CA 95141, USA.

Simon Laws, IBM United Kingdom Ltd, Hursley Park, Winchester S021 2JN, UK.

Gavin McCance, Department of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK.

James Magowan, IBM United Kingdom Ltd, Hursley Park, Winchester S021 2JN, UK.

Norman W. Paton, partment of Computer Science, University of Manchester, Oxford Road, Manchester M134 9PL, UK.

Greg Riccardi, Department of Computer Science, Florida State University, Tallahassee, FL 32306-4530, USA.

Trademarks

Glossary

Recommended by not required.

Intellectual Property Statement

Full Copyright Notice

References

M.P. Atkinson, M. Westhead, R. Baxter, N. Alpdemir, M. Antonioletti and S. Laws, Architectural Framework, OGSA-DAI Report EPCC-GDS-WP2-D2.1.0v0.3.5, Octobere, 2002.

W. H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. McCance and M. Silander, Project Spitfire – Towards Grid Web Service Databases, Presented at DAIS Working Group, GGF5, 2002.

S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, Internet Engineering Task Force, RFC 2119, http://www.ietf.org/rfc/rfc2119.txt, March 1997.

F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Schwchuk and T. Storey, Web Services Coordination (WS-Coordination), http://www-106.ibm.com/developerworks/library/ws-coor/, 2002-a.

F. Cabrera, G. Copeland, B. Fox, T. Freund, J. Klein, T. Storey and S. Thatte, Web Services Transaction (WS-Transaction), http://www.ibm.com/developerworks/library/ws-transpec/, 2002-b.

E. Christensen, F. Curbera, G. Meredith and S. Weerawanaa, Web Services Description Language (WSDL) 1.1, W3C Note, http://www.w3.org/TR/wsdl, W3C, 2001.

N.P. Chue Hong, A. Krause, S. Malaika, G. McCance, S. Laws, J. Magowan, N.W. Paton, G. Riccardi, Grid Database Server Specification Primer, In preparation, 2003.

B. Collins, A. Borley, N. Hardman, A. Knox, S. Laws, J. Magowan, M. Oevers, E. Zaluska, Grid Data Services – Relational Database Management Systems, Presented at GGF5, http://www.cs.man.ac.uk/grid-db, 2002.

D.C. Fallside, XML Schema Part 0: Primer, W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, W3C, 2001.

I. Foster, D. Gannon, The Open Grid Services Architecture Platform, Global Grid Forum Working Draft, 16th February 2003.

ISO/IEC (working draft) 9075-1 (SQL/Framework), ISO/IEC JTC 1/SC 32, 2002-01-11.
A. Krause, K. Smyllie and R. Baxter, Grid Data Service Specification for XML Databases, OGSA-DAI Report EPCC-GDS-WP3-XGDS 1.0, 2002.

N.W. Paton, M.P. Atkinson, V. Dialani, D. Pearson, T. Storey and P. Watson, Database Access and Integration Services on the Grid, Technical Report UkeS-2002-3, National e-Science Centre, 2002.

V. Raman, I. Narang, C. Crone, L. Haas, S. Malaika, T. Mukai, D. Wolfson and C. Baru, Data Access and Management Services on Grid, Presented at GGF5, http://www.cs.man.ac.uk/grid-db, 2002.

S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, P. Vanderpilt, Open Grid Services Infrastructure, Version 1.0, http://www.gridforum.org/ogsi-wg, March 13, 2003.