

Grid Database Service Specification

Document Identifier: GDSS-0.2

Date: 4th October 2002

Authors: 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)
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 (Department 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. + National e-Science Centre, 15 South College Street, Edinburgh EH8 9AA, UK)

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 artefacts 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 Database Services. The proposal is presented for discussion within the Global Grid Forum (GGF) Database Access and Integration Services (DAIS) Working Group, in the hope that it will evolve into a formal standard for Grid Database Services. 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.

Copyright (C) Global Grid Forum (4th October 2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."

Contents

Contents	2
1 Introduction	3
2 Overview	4
3 Generic Definitions	7
3.1 GridDataService PortType	7
3.1.1 GridDataService PortType: Service Data Descriptions and Elements	7
3.1.2 GridDataService PortType: Operations and Messages	10
3.1.3 GridDataService PortType: Types	10
3.2 GridDataTransport PortType.....	13
3.2.1 GridDataTransport PortType: Service Data Descriptions and Elements.....	15
3.2.2 GridDataTransport PortType: Operations and Messages.....	15
3.2.3 GridDataTransport PortType: Types and examples	16
4 Relational Database Services	25
4.1 GridDataService PortType	25
4.1.1 GridDataService PortType: Service Data Descriptions and Elements	25
4.1.2 GridDataService PortType: Operations and Messages	30
4.1.3 GridDataService PortType: Types	30
4.2 NotificationSource PortType.....	35
5 XML Database Services.....	35
5.1 GridDataService PortType	35
5.1.1 GridDataService PortType: Service Data Descriptions and Elements	35
5.1.2 GridDataService PortType: Operations and Messages	35
5.1.3 GridDataService PortType: Types	36
6 Remote Procedure Call.....	37
6.1 Generic Operations	37
6.2 Relational Database Specific.....	39
6.3 XML Database Specific.....	41
6.4 Implementation	41
6.5 Example	42
7 Conclusions	43
8 Acknowledgements	43
9 References.....	43

1 Introduction

This document presents a specification for a collection of Grid Database Services. The proposal is not *ab initio*, in that the following documents (at least) have been important in shaping our understanding of Grid Database Services [Atkinson 02, Bell 02, Collins 02, Krause 02, Paton 02].

The following principles have guided the development of the specification.

1. The specification is intended to provide service-based access to *existing* database 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 with a Grid or Web Services setting. The specification seeks to make as few assumptions as possible about the functionality, architecture, data model and language interfaces of databases that might be used within a Grid setting.
2. As there are several widely used database paradigms (e.g., relational, object, XML), the specification seeks to accommodate the different paradigms within a consistent framework. Thus those aspects of a service interface that are independent of the kind of database being accessed 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.
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 metadata to allow the service to be used given the specification of the service and the metadata provided by the service.
4. A Grid Database Service 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] is, or will evolve to be, sufficient and appropriate for characterising the transactional behaviour of services). Overall, the proposal seeks to accommodate the distinctive features of individual systems, while easing database access and integration activities within a Grid setting.
5. The Grid Database Service specification should be orthogonal to the Grid authentication and authorisation 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 database products define fine-grained access to the data they hold, for example, down to the column and row level for relational databases. A Grid Database Service implementation can choose the level of authorisation granularity exposed to the Grid users; the service does not seek to mandate this.
6. The specification adopts the document approach to service description. As such, there are relatively few portTypes, and most functionality is described using document definitions. This has the standard advantages (and disadvantages) of the document-based approach.
7. 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. It is intended that the OGSA-DAI project (http://umbriel.dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI/) will provide implementations and user documentation for these services in due course.

8. The specification seeks to support higher-level information-integration services. As such, features such as metadata description and data delivery support, have been designed with a view to providing the necessary ‘hooks’ for the developers of such services.

Some familiarity is assumed with XML Schema [Fallside 01], WSDL [Christensen 01] and the Open Grid Services Architecture (OGSA) [Tuecke 02] is assumed in what follows. This document is written in the context of Draft 3 (17th July 2002) of the OGSA Specification.

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY” AND “OPTIONAL” are to be interpreted as described in RFC2119 (<http://ietf.org/rfc/rfc2119.txt>).

2 Overview

In this specification, we describe the messages that flow between a client application (or library) and a data source, enabling clients to access or modify the content of data sources and their associated schemas. The portTypes that describe these messages define the interface to a Grid Database Service.

Although the specification is not prescriptive in this regard, each data source could be associated with many Grid Database Service instances, each representing a currently active client. In such a context, each client has its own unique relationship with the data source it is accessing expressed through the Grid Service Handle (GSH) [Tuecke 02] of the Grid Database Service instance. The GSH enables the application to locate a Grid Database Service and hence a data source. It is anticipated that a Grid Database Service is likely to map directly to a database session or connection of an underlying data management system. A common practice for large-scale systems is to share connections across many clients (called connection pulling). The same concept is likely to apply to Grid Database Services, but is transparent to the clients, and thus is not included explicitly in the specification provided here.

This section provides an overview of the specification, indicating its scope, and how this maps onto elements within the specification, which is presented in detail in the following sections. The following portTypes are specified or used:

- *GridDataService*. This is a new portType that provides functionality for accessing a database service [Atkinson 02]. The following operation is specified on *GridDataService*:
 - *GridDataService::perform*. This operation allows statements to be sent to the database, such as for query or update, and specifies how inputs to and results from such a request are handled.

As this operation may take and return complex documents, much of the functionality of a *GridDataService* is actually represented in the XML Schema definitions of the operands.

- *GridDataTransport*. This is a new portType that provides functionality for communicating results between networked resources. In the context of GridDatabaseServices, it is used to support flexible delivery of query results and flexible receipt of statement inputs. The following operation is specified on *GridDataTransport*:
 - *GridDataTransport::perform*.

As with *GridDataService*, the documents passed to and returned from this operation may be complex in nature.

- *GridService*. This is the mandatory portType for Grid Services that provides access to information about a service [Tuecke 02], which are described as Service Data Elements (SDEs). The specification provides XML Schema definitions for SDEs for

GridDataService, and as such provides information on the content and capabilities of the service.

- *NotificationSource*. This is the optional portType for Grid Services that enables a service to send notification messages [Tuecke 02]. The specification provides definitions for SDEs that describe the notification messages that a *GridDataService* may send.

It is intended that subsequent versions of the specification also include a portType that supports data translation/transformation. Such a portType should be relevant both for data being supplied to the Grid Database Service and to results supplied by the service.

The specification includes paradigm-dependent and paradigm-independent parts. The paradigm independent parts are presented first, and are shared by the specifications for relational and XML database services. The following are paradigm-independent:

- the names of the operations and various aspects of their parameters in the *GridDataService* portType;
- all aspects relating to the description and control of *GridDataTransport* – the data that is transported may be paradigm-dependent, but the description of how it is to be transported is paradigm independent; and
- certain of the SDEs of a service – for example, those relating to transport.

The following are paradigm-dependent:

- the language used to convey requests to a database service – for example, this could be SQL for relational database services and Xquery for XML database services;
- the types used to convey parameters into and results from the operations of the *GridDataService* portType; and
- the notification messages that may be generated by a database service.

We note that a single database system may support multiple paradigms – for example, relational vendors are increasingly providing storage and query facilities for XML data; this is accommodated by the specification.

The specifications in this document do not address discovery or creation of Grid Database Services directly. Thus it is assumed for now that a service can be discovered using an existing service registry, and that different practices may be pursued in the creation of Grid Database Service instances. To outline one possible approach, which is illustrated in Figure 1, a Grid Database Service instance could be created to represent a session over a database. Thus the service instance is created by a call to a factory, with a lifespan managed using the operations of the *GridService* portType [Tuecke 02]. In such a model, the service instance acts as a “proxy” for the database, as (at least for the meantime) data management systems do not support the portTypes described in this document directly.

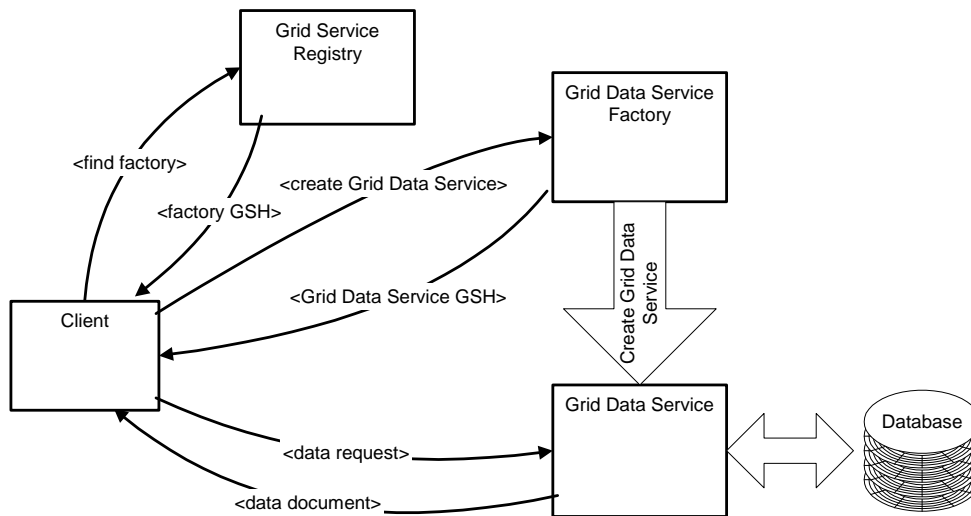


Figure 1. Creating and Using a Grid Data Service.

Once a service instance has been created or discovered, Figure 2 illustrates how the *GridDataService* portType introduced above can be used. The most straightforward usage pattern is that a query specification is sent from the requester to the Grid Data Service instance using *GridDataService::perform*, and the results of the query are returned directly to the requester as the operation result.

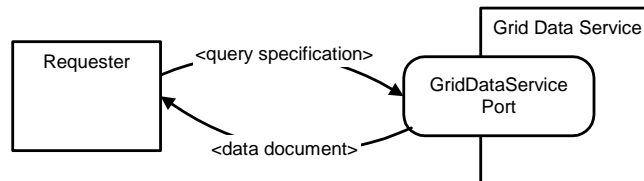


Figure 2. Requester Retrieving Data from a Grid Data Service.

In general, there are three parts to an interaction of a requester with a *GridDataService*, as illustrated in Figure 3. In the first part, the requester uses the *GridService* portType (e.g., by way of the *FindServiceData* operation) to access metadata on the service (e.g., relating to the schema of the database). If the requester already knows enough about the service to use it, this part can be omitted. In the second step, the *perform* operation of the *GridDataService* portType is used to convey a request to the *GridDataService*, for example, to evaluate a query. The results of the query could be returned to the requester directly, or a third step can take place. In the third step, the *perform* operation of the *GridDataTransport* portType is used to request delivery of the result of the query. This delivery could be to one or several destinations. In essence, this document makes a proposal that seeks to characterise the possible behaviours of each of these steps.

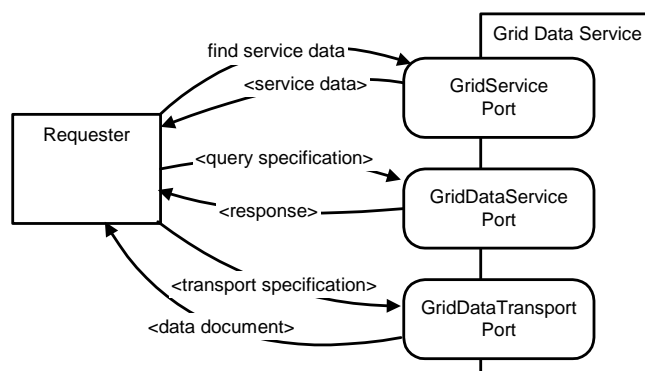


Figure 3. Requester Using Grid Data Service Ports.

The remainder of this document is structured as follows. Section 3 describes the two principal portTypes of this specification, namely *GridDataService* and *GridDataTransport*. In each case, following the structure of [Tuecke 02], service data descriptions and elements associated with each portType are described first, followed by descriptions of the operations supported by the portType. In Section 3, features associated with particular database models or languages are kept to a minimum. Sections 4 and 5 each cover the same ground as Section 3, but provide definitions for use with relational and XML databases, respectively. The portTypes described in Sections 3 to 5 are document-based, and thus sit at one end of a spectrum of possible approaches to service design. By contrast, Section 6 provides an RPC-based interface, whereby core functionalities can be accessed without the use of potentially complex documents. The two approaches can be considered to be complementary, in that users with straightforward requirements may find the RPC-based interface more straightforward to use. Section 7 presents some conclusions.

3 Generic Definitions

3.1 GridDataService PortType

The *GridDataService* portType is the principal context for database-specific operations and metadata. Although integrating database functionalities with a Grid middleware potentially involves integration of database capabilities with many other services (e.g., transactions, transport, security), most such functionalities are not specific to database access. Thus the *GridDataService* portType supports fairly rudimentary functionality, with an emphasis on request submission and result handling. However, making metadata available in a consistent manner is also considered important, and sufficient metadata SHOULD be made available by a service to allow its use given only the metadata and the documentation supplied with the service.

3.1.1 GridDataService PortType: Service Data Descriptions and Elements

The *GridDataService* portType is associated with *serviceData* elements conformant to the following *serviceDataDescription* elements:

```

<gsdl:serviceDataDescription
  name="LogicalSchema"
  type="xsd:anyType"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    An XML document that contains a description of the
    logical schema. This should be precise enough to
  
```

```
        allow creation of a corresponding schema at another
        location.
    </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="PhysicalSchema"
  type="xsd:anyType"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    An XML document that contains a description of the
    physical schema. This should be precise enough to allow
    higher-level services to allow a cost model to make
    predictions, for example, on query result sizes.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="StatementNotationTypes"
  type="xsd:anyURI"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    The list of notations that can be performed over the
    service.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="preparedStatements"
  type="xsd:anyURI"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    The identifiers of the currently prepared statements.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="resultCollections"
  type="xsd:anyURI"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    The identifiers of the currently available result
    collections.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="ResultFormatTypes"
  type="xsd:anyURI"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
```

```

        The list of query result formats that can be
        returned by the database.
    </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="DatabaseType"
  type="qname"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="constant">
  <wsdl:documentation>
    The type of database the service purports to be (e.g.
    relational, XML).
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="SystemName"
  type="qname"
  minOccurs="1"
  maxOccurs="1"
  mutability="mutable">
  <wsdl:documentation>
    The system that is used to provide the service.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="TransactionalCapability"
  type="xsd:anyType"
  minOccurs="1"
  maxOccurs="1"
  mutability="constant">
  <wsdl:documentation>
    The level of transaction support provided by the service.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

```

The *LogicalSchema* and *PhysicalSchema* SDEs provide information on the data model and its implemented representation, respectively. The XML schemas required to represent the database schemas are data model dependent, and thus defined separately for relational, object and XML databases.

There are also several SDEs that provide information on the kind of database supported by the service and its capabilities. The *StatementNotationTypes* SDE indicates the languages that can be used to direct requests to the service (e.g., an XML repository may support both Xquery and Xpath). The *DatabaseType* SDE indicates the kind of database service provided (for example, this document includes specifications for both relational and XML databases). This has a *maxOccurs* value of unbounded, as it is possible for a single database management system to provide multiple views of the data stored within it. For example, there is a move towards relational products being able to provide XML representations of their contents. In such a case, a single system would support multiple *DatabaseTypes*, with correspondingly many logical and physical schemas. The *SystemName* indicates which database platform is providing the service (e.g., Oracle, DB2).

The *TransactionalCapability* of a service indicates the willingness of a service to participate in (potentially distributed) transactions. It is assumed that Grid services will ultimately support the WS-Transaction proposal [Cabrera 02], and thus no distinct proposal is made here

for transaction-related portTypes. This document does not currently provide a model for describing the transactional properties of a service.

3.1.2 GridDataService PortType: Operations and Messages

GridDataService::perform

Perform a statement on a Grid Database Service.

Input:

- *gridDataServiceRequest*: The document that describes the request.

Output:

- *gridDataServiceResponse*: The document that provides the result.

Fault:

- The document that describes problems processing the request.

Every *GridDataService* MUST implement the *GridDataService::perform* operation. As the operation takes a document as a parameter, its behaviour is characterised principally by the documents that are taken as input and returned as results. These are described in the following subsection.

3.1.3 GridDataService PortType: Types

This section defines and describes the types of the parameters for *GridDataService::perform*.

A *dbStatement* captures a request phrased over a Grid Database Service. It associates a notation (e.g., a particular dialect of SQL) with a statement in that notation (e.g., a query or update in SQL). It also defined a formatting for the result of the statement (e.g. providing a document description for the bag of tuples returned by an SQL query).

```
<xsd:complexType name="dbStatement">
  <xsd:attribute name="notation" type="xsd:anyURI"
    use="required" />
  <xsd:attribute name="returnFormat" type="xsd:anyURI"
    use="required" />
  <xsd:attribute name="statementType">
    <simpleType>
      <restriction base="NMTOKEN">
        <enumeration value="query"/>
        <enumeration value="update"/>
        <enumeration value="procedureCall"/>
        <enumeration value="bulkLoad"/>
        <enumeration value="schemaUpdate"/>
      </restriction>
    </simpleType>
  <xsd:element name="expression" type="xsd:string"
    minOccurs="1" maxOccurs="1"/>
</xsd:complexType>
```

A *preparedStatement* associates a statement with an identifier by which it will subsequently be known. In general in databases, a statement can be prepared for execution (e.g., compiled and optimised) before it is actually run. Each time the statement is executed, which may be many times, it may be provided with parameters. The *terminationTime* indicates the time after which the Grid Database Service MAY remove the statement and any resources it is consuming. If no *terminationTime* is provided, it is assumed that the statement will continue to exist until the Grid Database Service instance is destroyed. As the lifespan of a Grid Database Service instance may often be quite short (e.g. representing a session over the database), this may be appropriate behaviour in many cases.

```

<xsd:complexType name="preparedStatement">
  <xsd:sequence>
    <xsd:element name = "dbStatement" type = "dbStatement"
      minOccurs="1" maxOccurs="1"/>
    <xsd:element name = "statementId" type = "xsd:anyURI"
      minOccurs="1" maxOccurs="1"/>
    <xsd:element name = "terminationTime" type = "xsd:dateTime"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

```

A *statementParameter* associates a *statementId* with parameters to the identified statement. Statement (and result) identifiers are represented here as having a type *xsd:anyURI*, but this is tentative. The *parameterValue* is of a type that differs from language to language, and thus which is specified in the sections on specific models.

```

<xsd:complexType name="statementParameter">
  <xsd:sequence>
    <xsd:element name = "parameterValue" type = "xsd:anyType"
      minOccurs="1" maxOccurs="1"/>
    <xsd:element name = "statementId" type = "xsd:anyURI"
      minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

```

An *executeStatement* captures which statement is to be executed. This can either be a given *dbStatement* or the identifier of a previously *preparedStatement*. The result of *executeStatement* is an XML representation of the result of the request. For example, in the case of an SQL query this may be a *RowSet*, as described in Section 4.1.3.

```

<xsd:complexType name="executeStatement">
  <choice>
    <xsd:element name = "statement" type = "dbStatement" />
    <xsd:element name = "statementId" type = "xsd:anyURI" />
  </choice>
</xsd:complexType>

```

An *executeStatementKeepResult* is similar to *executeStatement*, but instead of returning a result value, the result is stored internally by the service with the identifier *resultID*. This identifier can subsequently be used in requests to the *GridDataTransport* portType, as described in Section 3.2. The behaviour of the *terminationTime* is the same as for *preparedStatement*.

```

<xsd:complexType name="executeStatementKeepResult">
  <choice>
    <xsd:element name = "statement" type = "dbStatement" />
    <xsd:element name = "statementId" type = "xsd:anyURI" />
  </choice>
  <xsd:element name = "resultId" type = "xsd:anyURI" />
  <xsd:element name = "terminationTime" type = "xsd:dateTime"
    minOccurs="0" maxOccurs="1"/>
</xsd:complexType>

```

As both prepared statements and execution results can be identified and stored as state in a service instance, there needs to be a mechanism by which these can be disposed of. Furthermore, the *terminationTime* of identified artefacts may have to be changed. Such behaviour is supported by the following definition. Setting the *terminationTime* to 0 discards the identified artefact.

```

<xsd:complexType name="setTerminationTime">
  <xsd:element name = "identifier" type = "xsd:anyURI" />

```

```
<xsd:element name = "terminationTime" type = "xsd:dateTime">
</xsd:complexType>
```

A *statement* is any top-level statement. The *GridDataService::perform* operation takes as input a document that consists of one or more statements.

```
<xsd:complexType name="gridDataServiceRequest">
  <choice maxOccurs="unbounded">
    <xsd:element name = "executeStatement"
      type = "executeStatement" />
    <xsd:element name = "executeStatementKeepResult"
      type = "executeStatementKeepResult" />
    <xsd:element name = "preparedStatement"
      type = "preparedStatement" />
    <xsd:element name = "statementParameter"
      type = "statementParameter" />
    <xsd:element name = "setTerminationTime"
      type = "setTerminationTime" />
    <xsd:element name = "transportDescription"
      type = "gdts:gridTransportDescription" />
  </choice>
</xsd:complexType>
```

As an example of a document that could be used as a parameter to a *GridDataService::perform* operation, the following XML document, when performed, evaluates an SQL query over a database service, and returns the result directly to the requester.

```
<gridDataServiceRequest>
  <executeStatement>
    <dbStatement notation="http://www.gridforum.org/dais/lang/SQL92"
      returnFormat=
        "http://gridforum.org/dais/schema/webRowSet.xsd"
      statementType="query">
      <expression>select * from person where age = 21</expression>
    </dbStatement>
    <statementId>myStatement</statementId>
  </executeStatement>
</gridDataServiceRequest>
```

As a more comprehensive example, the following XML fragment provides a *gridDataServiceRequest* that contains three statements. The first such statement is a *preparedStatement*, which provides an SQL query to the database service, and associates this statement with the name *myStatement*. The “?” in the query string represents a parameter that has yet to be bound. This parameter is set by the second statement, which indicates that the first (and only) “?” is to be associated with the value *21*. The third statement executes *myStatement*. The prepared statement *myStatement* will continue to be available to subsequent requests to the Grid Database Service.

```
<gridDataServiceRequest>
  <preparedStatement>
    <dbStatement notation="http://www.gridforum.org/dais/lang/SQL92"
      returnFormat=
        "http://gridforum.org/dais/schema/webRowSet.xsd"
      statementType="query">
      <expression>select * from person where age = ?</expression>
    </dbStatement>
    <statementId>myStatement</statementId>
  </preparedStatement>
```

```

<statementParameter>
  <parameterValue>
    <SqlParameter position="1">
      <value>21</value>
    </SqlParameter>
  </parameterValue>
</statementParameter>

<executeStatement>
  <statementId>myStatement</statementId>
</executeStatement>
</gridDataServiceRequest>

```

The above definitions constitute the input to *GridDataService::perform*. The result of *perform* corresponds to the following XML Schema definition:

```

<xsd:complexType name="gridDataServiceResponse">
  <choice maxOccurs="unbounded">
    <xsd:element name="executeStatementResponse"
      type="xsd:anyType" />
    <xsd:element name="executeStatementKeepResultResponse"
      type="xsd:string" />
    <xsd:element name="preparedStatementResponse"
      type="xsd:string" />
    <xsd:element name="statementParameterResponse"
      type="xsd:string" />
    <xsd:element name="transportDescriptionResponse"
      type="gdts:gridTransportResponse" />
  </choice>
</xsd:complexType>

```

In essence, when a *gridDataService* request is performed by a *GridDataService*, each statement is performed in turn, and its response collected to yield a *gridDataServiceResponse*. When a statement fails, no subsequent statement is performed. An *executeStatementResponse* is the XML document that is the result of the request. An *executeStatementKeepResultResponse*, *preparedStatementResponse*, a *statementParameterResponse* or a *setTerminationTimeResponse* is either *ok* or an error report. The results of transport requests are described in Section 3.2.3.7.

3.2 GridDataTransport PortType

This section specifies and illustrates the receiving of data from and sending of data to a GDS. As with the *GridDataService* portType, each request to the *GridDataTransport* portType includes a document that specifies a change of state to the GDS and specifies a response to be returned to the requester. In most cases, *GridDataTransport* documents can be included in *GridDataService* documents and sent to the *GridDataService* port of a Grid Data Service.

The general strategy for using the *GridDataTransport* portType is to first specify one or more data queries or updates using *GridDataService* documents, and then to issue one or more transport requests. Transport requests to receive data can be used to send queried data directly to the requester, send data to one or more third parties, or distribute data among many third parties. Transport requests to send data to the GDS can be used for updating of data directly from the requester or from a third party.

Figure 4 and Figure 5 illustrate some of the possible interactions where data from a GDS is sent to one or more recipients. Each pair of arrows represents a single request and the response of the GDS to it.

In Figure 4, requester A sends a query specification document to the GDS. The response to that request is the identifier of the result of the query, as described in Section 3.1. The

requester subsequently sends a document containing a transport specification and receives the result data from the query in response.

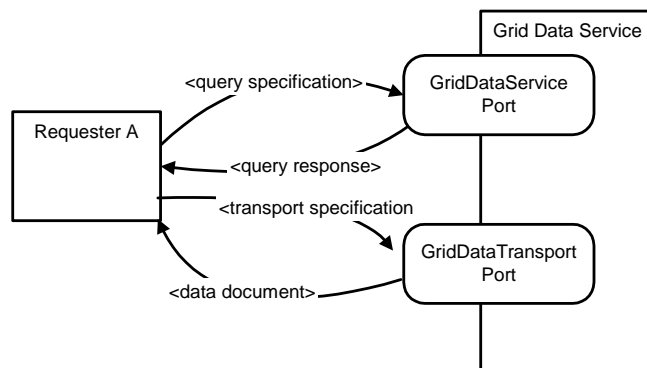


Figure 4. Direct query request and delivery to the requester

Figure 5 illustrates delivery of a single data set to multiple clients. Requester A first sends a query to the GDS, and then sends the resulting query identifier to requesters B and C. Finally B and C send documents to the GDS that request the transportation of the result data. The result data of the query is transmitted to both B and C in response to their requests.

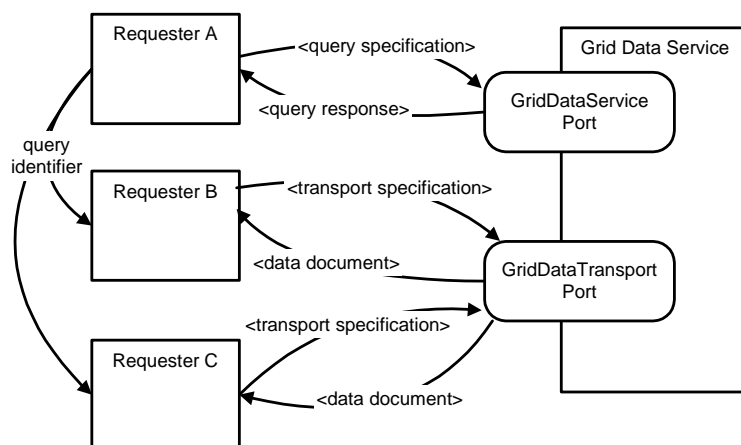


Figure 5. Query request with delivery to 2 third parties

For both receiving (*get*) and sending (*put*) data, the *GridDataTransport* portType supports *direct* and *indirect* operations. Block transfers (sequences of partial deliveries) are also supported for sending and receiving data.

The transport requests in Figure 4 and Figure 5 all represent requests to get data directly from the GDS. The response to a direct get request is the return of the appropriate data to the requester in the response document.

An indirect request to get data from a GDS identifies the data query source and the recipients. The recipients must be servers that are capable of delivering the data on request. The response to the *indirect* request is an acknowledgement to the requester and does not contain the requested data. The GDS will initiate the data transfer to the recipient servers.

For sending data (*put*) to the GDS, a *direct* request identifies the update statement that will receive the data and the values of the data. These data values are included in the request document.

An *indirect* request to put data into a GDS identifies the update statement and the source of the data. The data source must be a server that is capable of delivering the data on request. The response to the request is an acknowledgement. The GDS will initiate the data transfer from the source.

Block transfers allow clients to control the amount of data that moves to or from a GDS. A client can create a block transfer and then transfer the data as required. Some examples of block transfers are described in Section 3.2.3.5.

3.2.1 GridDataTransport PortType: Service Data Descriptions and Elements

The *GridDataTransport* portType is associated with *serviceData* elements conformant to the following *serviceDataDescription* elements:

```
<gsdl:serviceDataDescription
  name="gds:LogicallySupportedTypes"
  type="xsd:anyURI"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    An XML document that identifies the types of transport
    available from this service.
  </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
  name="gds:PhysicalPropertiesOfTypes"
  type="xsd:anyType"
  minOccurs="1"
  maxOccurs="unbounded"
  mutability="mutable">
  <wsdl:documentation>
    An XML document that describes the physical properties of
    the types of transport available from this GDS.
  </wsdl:documentation>
</gsdl:serviceDataDescription>
```

3.2.2 GridDataTransport PortType: Operations and Messages

As with the *GridDataService* portType, the *GridDataTransport* portType has a single operation.

GridDataService::perform

Perform a statement on a GDS.

Input:

- *GridDataTransportStatement*: The document that describes the request.

Output:

- *GridDataTransportResponse*: The document that provides the result.

Fault:

- *GridDatabaseTransportFault*: The document that provides the fault. This fault occurs when the GDS cannot process a request for any reason.

Every *GridDataTransportService* MUST implement the *GridDataTransportService::perform* operation. As the operation takes a document as a parameter, its behaviour is characterised

principally by the documents that are taken as input and returned as results. These are described in the following subsection.

Every *GridDataTransport* document must include parameters *direction* and *mode*, as described in Table 1. The value of parameter *direction* specifies whether data is travelling from the GDS (*get*) or toward the GDS (*put*). The value of parameter *mode* specifies how the data is to be moved. The two basic data movement strategies are values *direct* in which data is transferred directly to or from the client as part of the request or response document, and *indirect* in which data is transferred to or from a third-party server. Operations *block*, *directNext* and *indirectNext* support incremental movement of data in blocks in response to multiple requests.

Table 1. Required transport parameters and their meanings

Parameters		Meaning
direction	mode	
get	direct	Return the result of a query to the requester as part of the response document
	indirect	Send the result of a query to another server
	block	Setup a new block transfer for a query
	directNext	Return the next block of data from a block transfer as part of the response document
	indirectNext	Send the next block of data to another server
put	direct	Receive data that is included in the request document for an update
	indirect	Get data from another server for an update
	block	Setup a new block transfer to receive data for an update
	directNext	Receive the next block of data that is included in the request document for an update
	indirectNext	Get the next block of data from another server for an update

3.2.3 GridDataTransport PortType: Types and examples

Each call on the *perform* operation on a *GridDataTransport* portType must provide a *GridTransportDescription* document as input and will receive a *GridTransportResponse* document as its result. This section illustrates the roles that such documents play in representative scenarios.

3.2.3.1 Direct transfer of data from a GDS

The simplest *GridTransportDescription* documents are for direct get operations on query result data. Table 2 shows samples of a *gridDataServiceRequest* document (a), a *GridTransportDescription* document (b), and a *GridTransportResponse* document (c). The *GridTransportDescription* document of Table 2 (b) can be used for all of the requests for Figure 4 and Figure 5. In each case a single query request is followed by one or more transport requests. Each transport requests returns an identical copy of the data. The *maxSize* attribute of *GridTransportDescription* gives the maximum amount of data the requester is willing to accept. Argument *timeout* is similar. A value of 0 means no limit.

Table 2. *GridServiceDescription* and *GridTransportDescription* and *GridTransportResponse* documents for Figure 4 and Figure 5

(a) Query specification.

<pre> <gridDataServiceRequest> <executeStatementKeepResult> <dbStatement notation = "... " ...> <expression> select * from myData </expression> </dbStatement> <resultId> result1 </resultId> </executeStatementKeepResult> </gridDataServiceRequest> </pre>
(b) Direct get transport request for Figure 4 and Figure 5
<pre> <GridTransportDescription direction="get" mode="direct" maxSize="0" timeout="0"> <resultId> result1 </resultId> </GridTransportDescription> </pre>
(c) Response to transport request
<pre> <GridTransportResponse direction="get" mode="direct" maxSize="0" timeout="0"> <resultId> result1 </resultId> <ResultCollection> data returned in appropriate XML form </ResultCollection> </GridTransportResponse> </pre>

3.2.3.2 Indirect transfer of data from a GDS

An indirect operation to send data from a GDS to third-party servers is illustrated by Figure 6. Requester A specifies that data is to be sent to ftp servers B and C, as illustrated by the sample XML fragment in Table 3 (a). Once the GDS has determined that the data is available and can be sent to both targets, it sends requester A a *DataTransportResult* document like that in Table 3 (b). This result document has a *status* of *ok* to indicate that the transport operation could be initiated. It does not imply that the entire operation has been completed successfully.

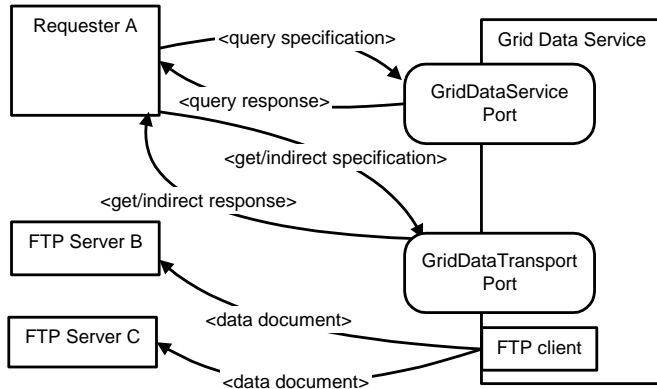


Figure 6. Indirect transfer of data to third-party servers

Table 3. *GridTransportDescription* and *GridTransportResponse* documents for Figure 6

(a) Indirect get specification for Figure 6.
<pre> <GridTransportDescription direction="get" mode="indirect"> <resultId> result1 </resultId> <TransportTarget protocol="ftp" target="B" file="data1"/> <TransportTarget protocol="ftp" target="C" file="data2"/> </GridTransportDescription> </pre>

(b) Result of transport request
<pre><GridTransportResponse direction="get" mode="indirect"> <resultId> statement1 </resultId> <TransportTarget protocol="ftp" target="B" file="data1" result="ok"/> <TransportTarget protocol="ftp" target="C" file="data2" result="ok"/> </GridTransportResponse></pre>

3.2.3.3 Direct transfer of update data to a GDS

A request to directly load data, as illustrated in Figure 7, begins with the specification of the update, as shown in Table 4 (a). The *bulkLoad* can be expected to name the portion of the database to be updated. Table 4 (b) shows an XML fragment that specifies the direct data transfer. The data to be loaded into the table is included in the document. The result document in Table 4 (c) has status *ok* to indicate that the update operation was completed successfully.

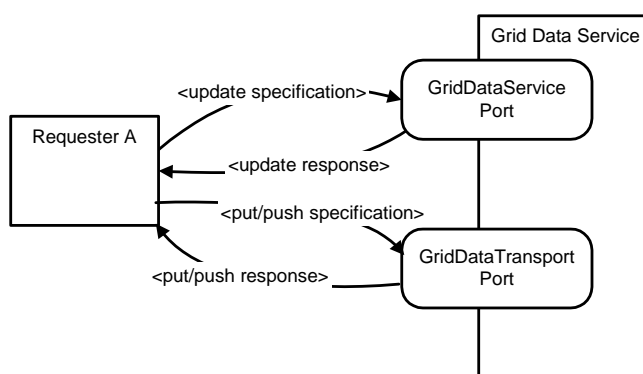


Figure 7. Direct delivery of update data to a GDS

Table 4. GridServiceDescription, GridTransportDescription, and GridTransportResponse documents for Figure 7.

(a) Update specification for Figure 7
<pre><preparedStatement> <dbStatement statementType="bulkLoad" notation = "..."> <expression> load table MyNewTable </expression> </dbStatement> <statementId> statement2 </statementId> </preparedStatement></pre>
(b) Direct put transport request for Figure 7
<pre><GridTransportDescription direction="put" mode="direct" maxSize="0"> <statementId> statement2 </statementId> <LoadTable> < ... data to be loaded in appropriate XML form/> </LoadTable> </GridTransportDescription></pre>
(c) Result of transport request
<pre><GridTransportResponse direction="put" mode="indirect" status="ok"> <statementId> statement2 </statementId> </GridTransportResponse></pre>

3.2.3.4 Indirect transfer of update data to a GDS

Figure 8 illustrates how data can be transferred from a third-party server to a GDS, and Table 5 shows sample XML fragments that specify the transfer. The requester *A* begins by creating a description of the *update* operation. Table 5 (a) shows the transport document that describes transferring a file from ftp server *B* to be stored in the table. The result document of Table 5 (b) has status *ok* to indicate that the update operation could be initiated. It does not imply that the entire operation has been completed successfully.

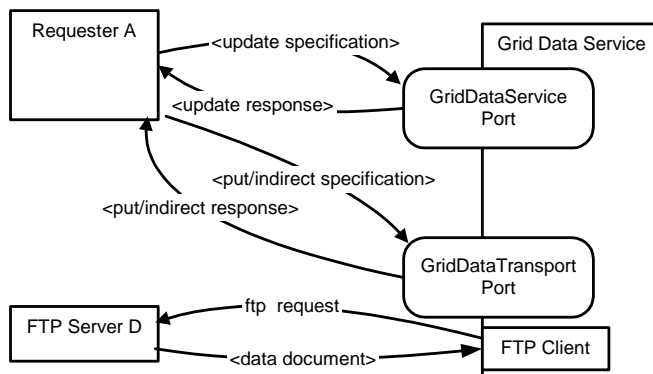


Figure 8. Indirect transfer of data from a third-party server

Table 5. *GridServiceDescription* and *GridTransportDescription* and *GridTransportResponse* documents for Figure 8

(b) Indirect put transport request for Figure 8.
<pre> <GridTransportDescription direction="put" mode="indirect" protocol="ftp" source="D" file="data" maxSize="0"> <statementId> statement2 </statementId> </GridTransportDescription> </pre>
(c) Result of transport request
<pre> <GridTransportResponse direction="put" mode="indirect" status="ok"> <statementId> statement2 </statementId> </GridTransportResponse> </pre>

3.2.3.5 Block transfer of query result data from a GDS

The *GridDataTransport* portType provides a facility for creating and using block transfers, as illustrated in Figure 9. In this case, requestor *A*, after specifying the query, requests the initialization of a block transfer. *A* then sends the block identifier to *B* who initiates a sequence of *directNext* requests from the GDS. Table 6 shows sample XML fragments for these operations.

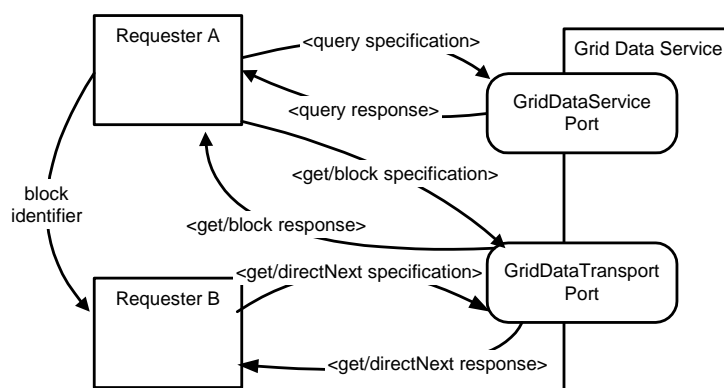


Figure 9. Direct block transfer of data

Table 6. *GridTransportDescriptions* and *GridTransportResponse* documents for iterating as in Figure 9

(a) Block transfer specification for Figure 9
<pre><GridTransportDescription direction="get" mode="block"> <resultId> result1 </resultId> <blockId> block1 </blockId> </GridTransportDescription></pre>
(b) Result of block request
<pre><GridTransportResponse direction="get" mode="block" status="ok"> <resultId> result1 </resultId> <blockId> block1 </blockId> </GridTransportResponse></pre>
(c) Direct request for next block of data from requester B
<pre><GridTransportDescription direction="get" mode="directNext" unit="rows" quantity="100"> <resultId> result1 </resultId> <blockId> block1 </blockId> </GridTransportDescription></pre>
(d) Result of <i>directNext</i> request
<pre><GridTransportResponse direction="get" mode="directNext" unit="rows" quantity="100" status="ok"> <resultId> result1 </resultId> <blockId> block1 </blockId> <ResultTable> 100 rows of data returned in appropriate XML form </ResultTable> </GridTransportResponse></pre>
(e) Result of final direct request
<pre><GridTransportResponse direction="get" mode="directNext" unit="rows" quantity="100" status="done"> <resultId> result1 </resultId> <blockId> block1 </blockId> <ResultTable> last rows of data returned in appropriate XML form </ResultTable> </GridTransportResponse></pre>

The requester continues to issue the transport requests until a result document like in Table 6 (e) is received. The *status* attribute with value *done* indicates that no more results are available.

Block transfers can be used to provide distribution and sharing of data, as illustrated in the XML fragments of Table 7. In Table 7 (a), requester *A* initiates 2 block transfers, named *block2* and *block3*. *A* then sends the identifier for *block2* to requesters *B* and *C* and sends identifier *block3* to *D*. When *B* and *C* request data using *block2*, they will be sent different rows of the result data. The values of *quantity* in Table 7 (b) and Table 7 (c) mean that each request from *B* returns 100 rows and each request from *C* returns 1000 rows. Once both *B* and *C* have finished their block transfers, the result data will have been distributed between them. The number of rows each receives will depend on how many requests each one makes. Requester *D*, however, is using independent block transfer *block3* and so will receive all of the result data of the query.

Table 7. *GridTransportDescriptions* and *GridTransportResponse* documents for sharing and distributing data

(a) Request from <i>A</i> to setup 2 block transfers
<pre><GridTransportDescription direction="get" mode="block"> <resultId> result1 </resultId> <blockId> block2 </blockId> </GridTransportDescription> <GridTransportDescription direction="get" mode="block"> <resultId> result1 </resultId> <blockId> block3 </blockId> </GridTransportDescription></pre>
(b) Distributed <i>directNext</i> get request for a block of data from requester <i>B</i>
<pre><GridTransportDescription direction="get" mode="directNext" units="rows" quantity="100" maxSize="0"> <resultId> result1 </resultId> <blockId> block2 </blockId> </GridTransportDescription></pre>
(c) Distributed <i>directNext</i> get request for a block of data from requester <i>C</i>
<pre><GridTransportDescription direction="get" mode="directNext" units="rows" quantity="1000" maxSize="0"> <resultId> result1 </resultId> <blockId> block2 </blockId> </GridTransportDescription></pre>
(d) Shared <i>directNext</i> get request for a block of data from requester <i>D</i>
<pre><GridTransportDescription direction="get" mode="directNext" units="rows" quantity="1000" maxSize="0"> <resultId> result1 </resultId> <blockId> block3 </blockId> </GridTransportDescription></pre>

Note that the distribution of data shown in Table 7 (b) and Table 7 (c) will only work properly when the blocks of data are semantically consistent. This example is of the distribution of data by rows to multiple processes. The processes must be capable of processing these partial results. It is unlikely that a distribution of data with *units="Kbytes"*, for example, will produce blocks of data that can be shared among multiple processes.

The other modes of block transfer are similar. Block transfers can be specified for both put and get directions and for indirect and direct modes.

3.2.3.6 Sending Data from One GDS to Another

The final example shows how to move data from one GDS to another. No new techniques are introduced. Instead, several service requests are combined to create the transfer of data. Figure 10 shows the basic architecture of the service requests used to send data from GDS *Y* to GDS *X*. Table 8 includes the sample documents that are used to create the data movement.

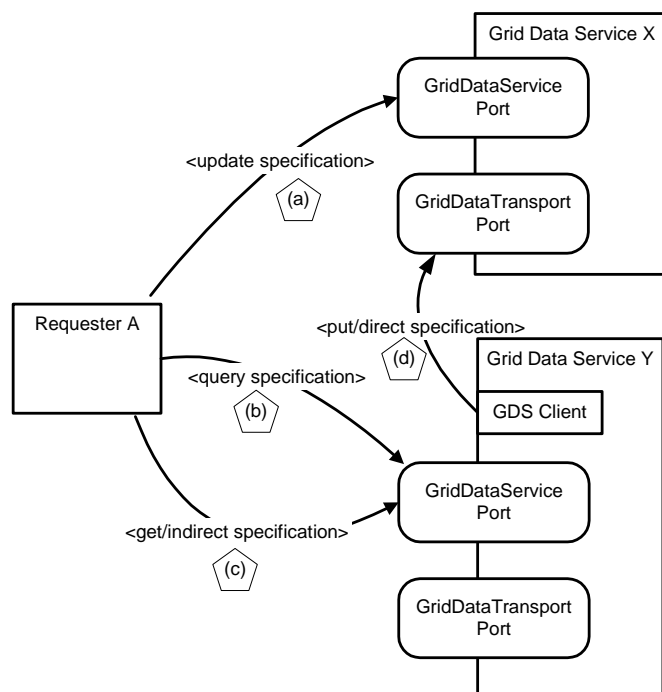


Figure 10. Sending data from one GDS to Another (responses omitted)

Table 8. GridServiceDescription and GridTransportDescription and GridTransportResponse documents for Figure 10.

(a) Bulk load of data
<pre><preparedStatement> <dbStatement statementType="bulkLod" notation = "..."...> <expression> load table MyNewTable </expression> </dbStatement> <statementId> statement3 </statementId> </preparedStatement></pre>
(b) Query of data to be moved
<pre><gridDataServiceRequest> <executeStatementKeepResult> <dbStatement notation = "..."...> <expression> select * from myData </expression> </dbStatement> <resultId> result4 </resultId> </executeStatementKeepResult> </gridDataServiceRequest></pre>
(c) Indirect request to transfer data to X
<pre><GridTransportDescription direction="get" mode="indirect"> <resultId> result4 </resultId> <TransportTarget protocol="GDS" target="X"> <statementId> statement3 </statementId> </TransportTarget> </GridTransportDescription></pre>
(d) Delivery of data to X: GDS Y acts as a client of X
<pre><GridTransportDescription direction="put" mode="indirect"> <statementId> statement3 </statementId> <LoadTable> data from result4 in appropriate XML form</pre>

```
</LoadTable>
</GridTransportDescription>
```

Four service documents are shown in Figure 10 and Table 8. The interaction begins when requester Request A sends document (a) to GDS X. Document (a) requests that X prepare to receive data and to store in the table called *MyNewTable*. This operation is labelled as *statement3*. The next step is for A to send document (b) to GDS Y. Document (b) requests that Y prepare to fetch the contents of table *MyData* and to label that query as *statement4*. After these two documents have been processed, X and Y are ready to accept data transport documents.

The third step is for A to tell Y to send the data to X using the GDS protocol. That is, document (c) tells Y to send data to X as a GDS client of X. GDS Y must extract the data from *statement4* and create document (d) as a request for a GDS to accept data to be delivered to update *statement3*. After creating the document, Y sends it to X and thus transfers the data. X receives the data that is included in document (d) and loads it into table *MyNewTable*. X can then send a response to Y to indicate a successful direct put, and Y can send a response to requester A to indicate a successful indirect get.

3.2.3.7 XML Schema Definitions

```
<xs:element name="GridTransportDescription">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="statementId" minOccurs="0">
      <xs:element ref="resultId" minOccurs="0">
      <xs:element ref="TransportTarget" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element ref="LoadTable" minOccurs="0"/>
      <xs:element ref="blockId" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="direction" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="get"/>
          <xs:enumeration value="put"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="mode" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="block"/>
          <xs:enumeration value="direct"/>
          <xs:enumeration value="directNext"/>
          <xs:enumeration value="indirect"/>
          <xs:enumeration value="indirectNext"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="maxSize" type="xs:int"/>
    <xs:attribute name="timeout" type="xs:int"/>
    <xs:attribute name="protocol" type="xs:string"/>
    <xs:attribute name="source" type="xs:string"/>
    <xs:attribute name="file" type="xs:string"/>
    <xs:attribute name="unit" type="xs:string"/>
    <xs:attribute name="quantity" type="xs:int"/>
  </xs:complexType>
</xs:element>
<xs:element name="GridTransportResponse">
```

```

<xs:complexType>
  <xs:sequence>
    <xs:element ref="statementId" minOccurs="0">
    <xs:element ref="resultId" minOccurs="0">
    <xs:element ref="blockId" minOccurs="0"/>
    <xs:element ref="ResultTable"/>
  </xs:sequence>
  <xs:attribute name="direction" type="xs:NMTOKEN"
    use="required"/>
  <xs:attribute name="mode" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="direct"/>
        <xs:enumeration value="indirectNext"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="unit" type="xs:string"/>
  <xs:attribute name="quantity" type="xs:int"/>
  <xs:attribute name="status" type="xs:string"/>
  <xs:attribute name="maxSize" type="xs:int"/>
  <xs:attribute name="timeout" type="xs:int"/>
</xs:complexType>
</xs:element>
<xs:element name="LoadTable" type="xs:string"/>
<xs:element name="ResultTable" type="xs:string"/>
<xs:element name="TransportTarget">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="statementId" minOccurs="0"/>
      <xs:element ref="resultId" minOccurs="0">
    </xs:sequence>
    <xs:attribute name="protocol" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="GDS"/>
          <xs:enumeration value="ftp"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="target" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="B"/>
          <xs:enumeration value="C"/>
          <xs:enumeration value="X"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="file">
      <xs:simpleType>
        <xs:restriction base="xs:NMTOKEN">
          <xs:enumeration value="data1"/>
          <xs:enumeration value="data2"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="setTerminationTime" minOccurs="0">
  <xs:complexType>

```

```

        <xsd:element name = "identifier" type = "xsd:anyURI" />
        <xsd:element name = "terminationTime"
            type = "xsd:dateTime">
    </xsd:complexType>
</xsd:element>
<xs:element name="blockId" type="xs:string"/>
<xs:element name="gridtransports">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="GridTransportDescription"/>
            <xs:element ref="GridTransportResponse"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="statementId" type="xs:anyURI"/>
<xs:element name="statementId" type="xs:anyURI"/>

```

4 Relational Database Services

4.1 GridDataService PortType

4.1.1 GridDataService PortType: Service Data Descriptions and Elements

A Relational Database Service MUST make available all service data elements defined in Section 3.1. Several of these, such as the *LogicalSchema* and *PhysicalSchema* include relational model-specific features, for which XML schema definitions are provided in this section.

4.1.1.1 LogicalSchema

A *RelationalDataService* MUST support the *LogicalSchema* SDE. The *LogicalSchema* SDE of a *RelationalDataService* contains an XML document describing the tables available within the Relational Database and the columns available for selection within those tables. It MAY also include primary key information and full datatypes for the columns.

Deleted: Primary Key

The root element of this document is a *databaseLogicalSchema*. A *databaseLogicalSchema* contains 1 or more tables, which in turn contain 1 or more columns. The following is an XML schema for the *databaseLogicalSchema* SDE:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="document" type="databaseLogicalSchema"/>
    <xs:complexType name="databaseLogicalSchema">
        <xs:sequence>
            <xs:element name="table" type="tableDefinition"
                minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="tableDefinition">
        <xs:sequence>
            <xs:element name="column" type="columnDefinition"
                minOccurs="1" maxOccurs="unbounded"/>
            <xs:element name="primaryKey" type="primaryKeyDefinition"
                minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="name" use="required" type="xs:string"/>
    </xs:complexType>

```

```
<xs:complexType name="columnDefinition">
  <xs:sequence>
    <xs:element name="sqlType" type="typeKeyword"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="xs:string"/>
  <xs:attribute name="fullName" use="required" type="xs:ID"/>
  <xs:attribute name="length" type="xs:integer"
    use="optional"/>
  <xs:attribute name="maxLength" type="xs:integer"
    use="optional"/>
  <xs:attribute name="characterSetName" type="xs:string"
    use="optional"/>
  <xs:attribute name="collation" type="xs:string"
    use="optional"/>
  <xs:attribute name="precision" type="xs:integer"
    use="optional"/>
  <xs:attribute name="scale" type="xs:integer" use="optional"/>
  <xs:attribute name="maxExponent" type="xs:integer"
    use="optional"/>
  <xs:attribute name="minExponent" type="xs:integer"
    use="optional"/>
  <xs:attribute name="userPrecision" type="xs:integer"
    use="optional"/>
  <xs:attribute name="leadingPrecision" type="xs:integer"
    use="optional"/>
  <xs:attribute name="maxElements" type="xs:integer"
    use="optional"/>
  <xs:attribute name="catalogName" type="xs:string"
    use="optional"/>
  <xs:attribute name="schemaName" type="xs:string"
    use="optional"/>
  <xs:attribute name="domainName" type="xs:string"
    use="optional"/>
  <xs:attribute name="typeName" type="xs:string"
    use="optional"/>
  <xs:attribute name="mappedType" type="xs:string"
    use="optional"/>
  <xs:attribute name="mappedElementType" type="xs:string"
    use="optional"/>
  <xs:attribute name="final" type="xs:boolean" use="optional"/>
</xs:complexType>

<xs:complexType name="primaryKeyDefinition">
  <xs:sequence>
    <xs:element name="columnFullName" type="xs:IDREF"
      minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="typeKeyword">
  <xs:restriction base="xs:string">
    <xs:enumeration value="CHAR"/>
    <xs:enumeration value="VARCHAR"/>
    <xs:enumeration value="CLOB"/>
    <xs:enumeration value="BLOB"/>
    <xs:enumeration value="NUMERIC"/>
    <xs:enumeration value="DECIMAL"/>
    <xs:enumeration value="INTEGER"/>
    <xs:enumeration value="SMALLINT"/>
    <xs:enumeration value="BIGINT"/>
    <xs:enumeration value="FLOAT"/>
  </xs:restriction>
</xs:simpleType>
```

```

    <xs:enumeration value="REAL" />
    <xs:enumeration value="DOUBLE PRECISION" />
    <xs:enumeration value="BOOLEAN" />
    <xs:enumeration value="DATE" />
    <xs:enumeration value="TIME" />
    <xs:enumeration value="TIME WITH TIME ZONE" />
    <xs:enumeration value="TIMESTAMP" />
    <xs:enumeration value="TIMESTAMP WITH TIME ZONE" />
    <xs:enumeration value="INTERVAL YEAR" />
    <xs:enumeration value="INTERVAL YEAR TO MONTH" />
    <xs:enumeration value="INTERVAL MONTH" />
    <xs:enumeration value="INTERVAL DAY" />
    <xs:enumeration value="INTERVAL DAY TO HOUR" />
    <xs:enumeration value="INTERVAL DAY TO MINUTE" />
    <xs:enumeration value="INTERVAL DAY TO SECOND" />
    <xs:enumeration value="INTERVAL HOUR" />
    <xs:enumeration value="INTERVAL HOUR TO MINUTE" />
    <xs:enumeration value="INTERVAL HOUR TO SECOND" />
    <xs:enumeration value="INTERVAL MINUTE" />
    <xs:enumeration value="INTERVAL MINUTE TO SECOND" />
    <xs:enumeration value="INTERVAL SECOND" />
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Even when one Grid Data Service relates to only one data source, databases can contain thousands of tables with thousands of columns. This means the *databaseLogicalSchema* defined above can become voluminous and volatile. Further Logical Schema information could be returned, this may include stored procedures, triggers etc. Schemas already exist for fully representing a databases logical structure in XML.

4.1.1.2 PhysicalSchema

The following describes the relational physical schema SDE, *databasePhysicalSchema*. There are two parts to this schema; the part describing the database size and the part describing the SQL capabilities of the database.

The schema for the database size part contains only two elements, database size and database free space, referring to the entire database backend.

The schema for the SQL capability list is taken from the ISO/IEC 9075 (SQL/Framework) standard [ISO 9075], section 6.3. It describes the variant of SQL supported by the *GridDataService*, the level of conformance to the standard and which optional parts and packages of the standard are supported. The parts refer to those discussed in documents ISO/IEC 9075-n, while the packages refer to those discussed in [ISO 9075] Appendix A. Refer to [ISO 9075] for discussion of the parts and packages of the SQL standard and the meaning of 'level of conformance'. Only a simple version of the schema is given here in XML; a service SHOULD implement all of the schema given here, and MAY implement further parts of the schema described in [ISO 9075].

```

<?xml version="1.0" encoding="UTF-8">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="document" type="databasePhysicalSchema">

  <xs:complexType name="databasePhysicalSchema">
    <xs:element name="databaseSize" type="sizeDef"
      minOccurs="1" maxOccurs="1"/>
    <xs:element name="SQLSupport" type="SQLSupportDef"
      minOccurs="1" maxOccurs="1"/>
  </xs:complexType>

```

```
<xs:complexType name="sizeDef">
  <xs:element name="sizeBytes" type="xs:int"
    use="required">
  <xs:element name="freeSpaceBytes" type="xs:int"
    use="required">
</xs:complexType>

<xs:complexType name="SQLSupportDef">
  <xs:element name="sql-variant" type="variantDef"
    minOccurs="1" maxOccurs="1">
</xs:complexType>

<xs:complexType name="variantDef">
  <xs:sequence>
    <xs:element name="sql-edition" type="editionDef"
      minOccurs="1" maxOccurs="1">
    <xs:element name="sql-conformance"
      type="conformanceDef"
      minOccurs="1" maxOccurs="1">
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="editionDef">
  <restriction base="NMTOKEN">
    <enumeration value="1992">
    <enumeration value="1999">
  </restriction>
</xs:simpleType>

<xs:complexType name="conformanceDef">
  <xs:sequence>
    <xs:element name="level" type="levelDef"
      minOccurs="0" maxOccurs="1">
    <xs:element name="parts" type="partsDef"
      minOccurs="0" maxOccurs="1">
    <xs:element name="packages" type="packagesDef"
      minOccurs="0" maxOccurs="1">
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="levelDef">
  <restriction base="NMTOKEN">
    <enumeration value="low">
    <enumeration value="intermediate">
    <enumeration value="high">
  </restriction>
</xs:simpleType>

<xs:complexType name="partDef">
  <xs:sequence>
    <xs:element name="part1" type="partNDef"
      minOccurs="1" maxOccurs="1">
      :
      :
    <xs:element name="partN" type="partNDef"
      minOccurs="1" maxOccurs="1">
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="PartNDef">
  <restriction base="NMTOKEN">
```

```

        <enumeration value="yes">
        <enumeration value="no">
    </restriction>
</xs:simpleType>

<xs:complexType name="packageDef">
    <xs:sequence>
        <xs:element name="packageI" type="partNDef"
            minOccurs="1" maxOccurs="1">
            :
            :
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="PackageIDef">
    <restriction base="NMTOKEN">
        <enumeration value="yes">
        <enumeration value="no">
    </restriction>
</xs:simpleType>

```

An example of the relational physical schema SDE is given below:

```

<databasePhysicalSchema>
  <databaseSize>
    <sizeBytes>13832145</sizeBytes>
    <freeSpaceBytes>195730041</freeSpaceBytes>
  </databaseSize>
  <SQLSupport>
    <sql-variant>
      <sql-edition>1999</sql-edition>
      <sql-conformance>
        <parts>
          <part1>yes</part1>
          <part1>yes</part1>
          <part2>yes</part2>
          <part3>yes</part3>
          <part4>yes</part4>
          <part5>yes</part5>
          <part6>yes</part6>
          <part7>yes</part7>
          <part8>no</part8>
          <part9>no</part9>
          <part10>no</part10>
          <part11>no</part11>
          <part12>no</part12>
          <part13>no</part13>
          <part14>yes</part14>
        </parts>
        <packages>
          <package001>yes</package001>
          <package002>yes</package002>
          <package003>no</package003>
          <package004>no</package004>
          <package005>no</package005>
          <package006>no</package006>
          <package007>no</package007>
          <package008>no</package008>
          <package009>no</package009>
        </packages>
      </sql-conformance>
    </sql-variant>
  </SQLSupport>
</databasePhysicalSchema>

```

```

        </sql-variant>
    </SQLSupport>
</databasePhysicalSchema>

```

4.1.2 GridDataService PortType: Operations and Messages

A Relational Database Service MUST provide the operations and messages defined in the generic section of this document. It does not support any additional operations and messages.

4.1.3 GridDataService PortType: Types

Each call on the *GridDataService::perform* operation must provide a *gridDataServiceRequest* document as input and will receive a *gridDataServiceResponse* document as its result.

Although the most common query language for relational databases is likely to be SQL, the standard permits services support others, for example XQuery and specific versions of SQL; the language is specified in the *notation* attribute of *dbStatement* in Section 3.1.3.

We will assume that the result of an SQL query to a GridDataService is an XML document (complete or a fragment) containing the result set or partial result set. The standard permits services to support many return formats for this document, specified by the *returnFormat* attribute of *dbStatement*. For the moment, the only recommended format for relational databases is based upon the Java WebRowSet (yet to be accepted into Java base). This has methods to convert a result set into XML and this does not appear to be XML specific. Schema for results from SQL query:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="RowSet">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="properties"/>
        <xsd:element ref="metadata"/>
        <xsd:element ref="data"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="properties">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="command"/>
        <xsd:element ref="concurrency"/>
        <xsd:element ref="datasource"/>
        <xsd:element ref="escape-processing"/>
        <xsd:element ref="fetch-direction"/>
        <xsd:element ref="fetch-size"/>
        <xsd:element ref="isolation-level"/>
        <xsd:element ref="key-columns"/>
        <xsd:element ref="map"/>
        <xsd:element ref="max-field-size"/>
        <xsd:element ref="max-rows"/>
        <xsd:element ref="query-timeout"/>
        <xsd:element ref="read-only"/>
        <xsd:element ref="rowset-type"/>
        <xsd:element ref="show-deleted"/>
        <xsd:element ref="table-name"/>
        <xsd:element ref="url"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="command" type="xsd:string"/>
  <xsd:element name="concurrency" type="xsd:string"/>

```

```

<xsd:element name="datasource" type="xsd:string"/>
<xsd:element name="escape-processing" type="xsd:string"/>
<xsd:element name="fetch-direction" type="xsd:string"/>
<xsd:element name="fetch-size" type="xsd:string"/>
<xsd:element name="isolation-level" type="xsd:string"/>
<xsd:element name="key-columns">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0"
        ref="column"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="column" type="xsd:string"/>
<xsd:element name="map">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded" minOccurs="0">
      <xsd:element ref="type"/>
      <xsd:element ref="class"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="type" type="xsd:string"/>
<xsd:element name="class" type="xsd:string"/>
<xsd:element name="max-field-size" type="xsd:string"/>
<xsd:element name="max-rows" type="xsd:string"/>
<xsd:element name="query-timeout" type="xsd:string"/>
<xsd:element name="read-only" type="xsd:string"/>
<xsd:element name="rowset-type" type="xsd:string"/>
<xsd:element name="show-deleted" type="xsd:string"/>
<xsd:element name="table-name" type="xsd:string"/>
<xsd:element name="url" type="xsd:string"/>
<xsd:element name="metadata">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="column-count"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0"
        ref="column-definition"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="column-definition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="column-index"/>
      <xsd:element ref="auto-increment"/>
      <xsd:element ref="case-sensitive"/>
      <xsd:element ref="currency"/>
      <xsd:element ref="nullable"/>
      <xsd:element ref="signed"/>
      <xsd:element ref="searchable"/>
      <xsd:element ref="column-display-size"/>
      <xsd:element ref="column-label"/>
      <xsd:element ref="column-name"/>
      <xsd:element ref="schema-name"/>
      <xsd:element ref="column-precision"/>
      <xsd:element ref="column-scale"/>
      <xsd:element ref="table-name"/>
      <xsd:element ref="catalog-name"/>
      <xsd:element ref="column-type"/>
      <xsd:element ref="column-type-name"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="column-count" type="xsd:string"/>
<xsd:element name="column-index" type="xsd:string"/>
<xsd:element name="auto-increment" type="xsd:string"/>
<xsd:element name="case-sensitive" type="xsd:string"/>
<xsd:element name="currency" type="xsd:string"/>
<xsd:element name="nullable" type="xsd:string"/>
<xsd:element name="signed" type="xsd:string"/>
<xsd:element name="searchable" type="xsd:string"/>
<xsd:element name="column-display-size" type="xsd:string"/>
<xsd:element name="column-label" type="xsd:string"/>
<xsd:element name="column-name" type="xsd:string"/>
<xsd:element name="schema-name" type="xsd:string"/>
<xsd:element name="column-precision" type="xsd:string"/>
<xsd:element name="column-scale" type="xsd:string"/>
<xsd:element name="catalog-name" type="xsd:string"/>
<xsd:element name="column-type" type="xsd:string"/>
<xsd:element name="column-type-name" type="xsd:string"/>
<xsd:element name="data">
    <xsd:complexType>
        <xsd:sequence maxOccurs="unbounded" minOccurs="0">
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                ref="row"/>
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                ref="ins"/>
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                ref="del"/>
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                ref="insdel"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="row">
    <xsd:complexType>
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element ref="col"/>
            <xsd:element ref="upd"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ins">
    <xsd:complexType>
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element ref="col"/>
            <xsd:element ref="upd"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name="del">
    <xsd:complexType>
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element ref="col"/>
            <xsd:element ref="upd"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name="insdel">
    <xsd:complexType>
        <xsd:choice maxOccurs="unbounded" minOccurs="0">

```

```

        <xsd:element ref="col"/>
        <xsd:element ref="upd"/>
    </xsd:choice>
</xsd:complexType>
</xsd:element>
<xsd:element name="col" type="xsd:string"/>
<xsd:element name="upd" type="xsd:string"/>
</xsd:schema>

```

A client MAY specify a *GridDataTransport* description with the query. The default behaviour of the data service is direct delivery (*pull* as described in Section 3.2). The data service MAY support the *GridDataTransport* port type providing other options like iteration or distributed delivery.

The following example shows a relational request formatted in SQL 92. The example given is of a simple customer database, with four fields requested, name, address, age and cross-reference index. It makes no reference to transport, so the default, direct pull is assumed by the service.

```

<gridDataServiceRequest>
  <preparedStatement>
    <dbStatement notation="http://www.gridforum.org/dais/lang/SQL92"
      returnFormat=
        "http://gridforum.org/dais/schema/webRowSet.xsd"
      statementType="query">
      <expression>select name, address, age, numericalindex
        from Customer where totalDebt > ?</expression>
    </dbStatement>
    <statementId>bountyHunt</statementId>
  </preparedStatement>

  <statementParameter>
    <parameterValue>
      <SQLParameter position="1">
        <value>5000</value>
      </SQLParameter>
    </parameterValue>
  </statementParameter>

  <executeStatement>
    <statementId>bountyHunt</statementId>
  </executeStatement>
</gridDataServiceRequest>

```

Deleted: <http://www.gridforum.org/dais/lang/SQL92>

The following shows are sample document returned from the above query. For brevity, not all of the defined elements in the *WebRowSet* schema are included here (these can be assumed to be empty).

```

<RowSet>
  <properties>
    <key-columns>
      <column>Name</column>
      <column>NumericalIndex</column>
    </key-columns>
    <read-only>true</read-only>
    <max-rows>1000</max-rows>
    <url></url>
  </properties>

  <metadata>
    <column-count>3</column-count>
    <column-definition>

```

```
        <column-index>1</column-index>
        <auto-increment>false</auto-increment>
        <nullable>false</nullable>
        <case-sensitive>true</case-sensitive>
        <column-name>Name</column-name>
        <column-scale>100</column-scale>
        <column-label>Customer name</column-label>
        <tablename>Customer</tablename>
        <column-type>xsd:string</column-type>
    </column-definition>
    <column-definition>
        <column-index>2</column-index>
        <auto-increment>false</auto-increment>
        <nullable>false</nullable>
        <case-sensitive>true</case-sensitive>
        <column-name>Address</column-name>
        <column-scale>200</column-scale>
        <column-label>Customer's Address</column-label>
        <tablename>Customer</tablename>
        <column-type>xsd:string</column-type>
    </column-definition>
    <column-definition>
        <column-index>3</column-index>
        <auto-increment>false</auto-increment>
        <nullable>true</nullable>
        <column-name>Age</column-name>
        <column-precision>1</column-precision>
        <column-label>Customer's age</column-label>
        <tablename>Customer</tablename>
        <column-type>xsd:integer</column-type>
    </column-definition>
    <column-definition>
        <column-index>4</column-index>
        <auto-increment>true</auto-increment>
        <nullable>false</nullable>
        <column-name>NumericalIndex</column-name>
        <column-precision>1</column-precision>
        <column-label>Quick index</column-label>
        <tablename>Customer</tablename>
        <column-type>xsd:integer</column-type>
    </column-definition>
</metadata>
<data>
    <row>
        <col>Gavin McCance</col>
        <col>Gav's place, Glasgow</col>
        <col>26</col>
        <col>1</col>
    </row>
    <row>
        <col>James Magowan</col>
        <col>James' house</col>
        <col>null</col>
        <col>2</col>
    </row>
</data>
</RowSet>
```

4.2 NotificationSource PortType

It is recommended that a Relational Grid Data Service allow notification of changes in its Service Data Elements. A client may subscribe to being notified about changes in an SDE using *subscribeByServiceDataName* [Tuecke 02].

Relational databases provide triggering capability and it is not yet clear how or if this should be exposed through the *GridDataService*. This would allow not only notification based on changes to data items that are not necessarily Service Data Elements, building on the triggering facilities of the database.

5 XML Database Services

5.1 GridDataService PortType

5.1.1 GridDataService PortType: Service Data Descriptions and Elements

A XML Database Service MUST make available all service data elements defined in Section 3.

5.1.1.1 LogicalSchema

The *LogicalSchema* SDE of an XML Database Service contains an XML document describing the hierarchy of the collections in the database. It MAY also provide XML schemas or DTDs for the documents stored in a collection. Note that documents in an XML collection do not need to satisfy a schema.

The root element of this document is a collection. A collection can contain 0 or more collections and 0 or more XML schemas. The following is an XML schema for the *LogicalSchema* SDE:

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
  <xs:element name="document" type="collectionType"/>
  <xs:complexType name="collectionType">
    <xs:sequence>
      <xs:element name="collection" type="collectionType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="schema" type="xs:anyType"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

5.1.1.2 PhysicalSchema

As there are no standards or conventions for physical schemas yet in XML databases, we make some suggestions for possible *PhysicalSchema* content:

- Number of collections.
- Number of schemas in a collection.
- Number of documents in a collection.
- Location and nature of indexes.

5.1.2 GridDataService PortType: Operations and Messages

An XML database service MUST provide the operations and messages defined in Section 3.1 of this document. It does not support any additional operations and messages.

5.1.3 GridDataService PortType: Types

Each call on the *GridDataService::perform* operation must provide a *gridDataServiceRequest* document as input and will receive a *gridDataServiceResponse* document as its result. The type of the result document will depend on the language of the request and the delivery requirements.

5.1.3.1 XPath

The XPath 2.0 working draft from 16th August 2002 states that the value of an XPath expression is always a sequence, which is an ordered collection of zero or more items. An item is either an atomic value or a node (see <http://www.w3.org/TR/xpath20/>, section 2: Basics).

We will assume that the result of an XPath query to a *GridDataService* is an XML document (complete or a fragment) containing a sequence of serialised items.

A client MAY specify a *GridDataTransport* description with the query. The default behaviour of the data service is direct delivery (direction="get" and mode="direct" as described in Section 3.2). The data service MAY support the *GridDataTransport* port type, providing other options like iteration or distributed delivery.

Deleted: operation

The following shows a sample document for a simple query to an XML data service. Consider a mail repository containing XML documents like this:

```
<posting>
  <to>Bob</to>
  <from>Alice</from>
  <subject>Example</subject>
  <body>This is an example for a document</body>
</posting>
```

Here is the document for a single query request using the XPath query language. The data will be returned directly to the client.

```
<gridDataServiceRequest>
  <executeStatement>
    <statement
      notation="http://www.w3.org/TR/1999/REC-xpath-19991116"
      statementType="query">
      <expression>//posting/to[text()='Bob']</expression>
    </statement>
  </executeStatement>
</gridDataServiceRequest>
```

The result document of this query is:

```
<gridDataServiceResponse>
  <executeStatementResponse>
    <result xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      type="xsi:any">
      <posting>
        <to>Bob</to>
        <from>Alice</from>
        <subject>Example</subject>
        <body>This is an example for a document.</body>
      </posting>
    </result>
    <result>
      ...
    </result>
  </executeStatementResponse>
</gridDataServiceResponse>
```

5.1.3.2 XQuery

The XQuery 1.0 working draft from 16th August 2002 states that the result of an XQuery request is a sequence. A sequence is an ordered collection of zero or more items. An item may be a node or a simple value (See <http://www.w3.org/TR/xquery/>, section 2: Basics).

We will assume that the result of an XQuery request to a *GridDataService* is a XML document containing a sequence of serialized items. A client MAY include the delivery description in the XQuery string or MAY append a *GridDataTransport* description to the operation.

6 Remote Procedure Call

Although the primary interface uses the web services document model, there are a number of cases for which the web services Remote Procedure Call (RPC) model is appropriate [Bell 02]. While the rigidity of RPC encoding limits flexibility, it provides a number of advantages for the ease of implementation and use, chief among these being the possibility of dynamic creation of client stubs from the service definition, and the ability of client code to link against these RPC stubs in a straightforward manner. The transport and encoding is not defined here, however the definition is given so as to be suitable for the standard web-services encoding, SOAP-RPC.

Deleted: num ber

The RPC operations defined here are synchronous, i.e. the direct transport is used with the result being returned to the client immediately. Transactional support and session support are not discussed; these are left for the next version of the RPC interface. The purpose here is to show how the RPC operations are related to the document approach of the generalised *GridDataService*. A *GridDataService* MAY implement one or more of the following operations. Some operations are generic; others are specific to the database type. Upon error these operations MUST throw a fault, e.g. a SOAPFault in the SOAP-RPC case. It is assumed that all operations here will throw a fault in the case of an authentication or authorisation failure.

6.1 Generic Operations

GridDataService::ShowDatabases

Return the names of all the databases held by the service.

Input

- *None*.

Output

- *DatabaseList*: The list of databases held by the service. This SHOULD be formatted as XML.

Fault

- *None*.

GridDataService::ExecuteUpdate

Execute a database update using the specified query notation.

Input

- *Notation*: The query notation for the update request. A database MAY support many update notations. Currently defined notations are specified using a URI.
 1. SQL 92: “<http://www.gridforum.org/dais/lang/SQL92>”
 2. XPath 1.0: “<http://www.w3.org/TR/1999/REC-xpath-19991116>”

3. XQuery 1.0: “http://www.w3.org/TR/2002/WD-xquery-20020816”

- *DatabaseName*: The database to update.
- *UpdateRequest*: The update string describing the request.

Output

- *UpdateResult*: The result of the update. This SHOULD be an integer describing how many database elements were affected (e.g. rows in the case of relational databases). It SHOULD be formatted in XML.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The request could not be parsed.
- *InvalidOperation*: The request failed for some reason (e.g. database does not exist). The fault SHOULD contain specifying the exact reason for the failure, subject to security policies.

GridDataService::ExecuteSchemaUpdate

Execute a database schema update in the specified query notation.

Input

- *Notation*: The query notation for the schema update request. A database MAY support many schema update notations. The currently defined notation is SQL, specified using the URI above.
- *DatabaseName*: The database to update.
- *SchemaUpdateRequest*: The schema update string describing the request.

Output

- *SchemaUpdateResult*: The result of the schema update MAY be returned. This is an integer describing how many schema items were affected (e.g. tables in the case of relational databases). If present, it should be formatted in XML.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The request could not be parsed.
- *InvalidOperation*: The request failed for some reason (e.g. database does not exist). The fault SHOULD contain a message specifying the reason for the failure, subject to security policies. The response of the database should be encoded in this message, for example, the SQLSTATE and SQLCODE for SQL database errors.

GridDataService::ExecuteQuery

Will execute a database query using the specified query notation.

Input

- *Notation*: The query notation for the request. A database MAY support many query notations. Currently defined notations are “SQL”, “XPath” and “XQuery”, specified using the URIs above.
- *DatabaseName*: The database to query.
- *Query*: The query string.

- *Maximum*: The maximum number of data units to return. The data unit is defined by the database type.

Output

- *QueryResult*: The result of the query. The full result should be given, up to the maximum specified. This SHOULD be in a suitable XML format, for example, an XML *WebRowSet* for SQL requests.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The query could not be parsed.
- *InvalidOperation*: The query failed for some reason (e.g. database does not exist). The fault SHOULD contain specifying the exact reason for the failure, subject to security policies.

6.2 Relational Database Specific

Most of these return information concerning the schema of the relational database.

GridDataService::ShowTables

Return the names of the tables held in a specific database. This operation is likely to need refining (including, for example, LIKE clauses) to limit the number of table names returned.

Input

- *DatabaseName*: The database name whose tables are to be listed.

Output

- *TableList*: The list of tables in the database. This SHOULD be formatted in XML.

Fault

- *NonexistentDb*: The specified database does not exist.

GridDataService::ShowTableColumns

Return the column names defined for a specific table. It is likely that the following specific operations will be combined with the *ShowTables* operation at a later date.

Input

- *DatabaseName*: The database name.
- *TableName*: The table whose columns are to be listed.

Output

- *ColumnList*: The list of columns in the database. This SHOULD be formatted in XML.

Fault

- *NonexistentDb*: The specified database does not exist.
- *NonexistentTable*: The specified table does not exist in the given database.

GridDataService::ShowTableColumnTypes

Returns the types of the columns for a specific table.

Input

- *DatabaseName*: The database name.
- *TableName*: The table whose columns are to be listed.

Output

- *ColumnTypeList*: The list of types of the columns in the database. This SHOULD be formatted in XML.

Fault

- *NonexistentDb*: The specified database does not exist.
- *NonexistentTable*: The specified table does not exist in the given database.

GridDataService::ShowFullSchema

Returns the full schema of the table or view requested, including all the stored triggers and indices.

Input

- *DatabaseName*: The database name.
- *TableName*: The table whose schema is to be returned.

Output

- *TableSchema*: The full schema describing the table. This should be formatted in XML.

Fault

- *NonexistentDb*: The specified database does not exist.
- *NonexistentTable*: The specified table does not exist in the given database.

GridDataService::bulkLoad

Inserts the *bulkData* into the specified table. This operation is appropriate for small loads that can be reasonably accomplished in one transaction. Further work needs to be done on the interface to permit more flexible bulk loading and error recovery. For larger loads, the approach illustrated in Section 3.2.3.4 is more appropriate.

Input

- *DatabaseName*: The database name.
- *TableName*: The table into which to insert.
- *BulkData*: The data to insert. The format of BulkData is in the XML *WebRowSet* schema.

Output

- *RowsInserted*: The number of rows successfully inserted. It SHOULD be formatted in XML.

Fault

- *NonexistentDb*: The specified database does not exist.
- *NonexistentTable*: The specified table does not exist in the given database.
- *InvalidBulkDataFormat*: The bulk data is not in the correct XML format.

- *SchemaMismatch*: The schema of the bulk data and the schema of the table being inserted into do not match.

6.3 XML Database Specific

GridDataService::ShowCollections

Returns the names of collections in the database.

Input

- *None*.

Output

- *CollectionList*: The list of collections in the database. This SHOULD be formatted in XML.

Fault

- *None*.

GridDataService::ShowDocuments

Returns the names of the documents held in a specific database.

Input

- *Collection*: The collection name whose documents are to be listed.

Output

- *DocumentList*: The list of documents in the collection. This SHOULD be formatted in XML.

Fault

- *NonexistentCollection*: The specified collection does not exist.

GridDataService::BulkLoad

Inserts an XML document into the specified collection. This operation is appropriate for small loads that can be reasonably accomplished in one transaction. For larger loads, the approach illustrated in Section 3.2.3.4 is more appropriate.

Input

- *Collection*: The collection into which the document will be inserted.
- *Document*: The document to insert.

Output

- *None*.

Fault

- *NonexistentCollection*: The specified collection does not exist.
- *InvalidDocumentFormat*: The document is not formatted correctly according to its schema.
- *SchemaMismatch*: The document is not of the appropriate schema for this collection.

6.4 Implementation

For the web-services SOAP transport, all these operation calls, parameters and returned results are encoded in an XML document using the SOAP-RPC schema. The preceding sections in this document describe the standard for the XML *GridDataService* schema. Since

all the possible operations of the above RPC requests are simply a subset of the generalised *GridDataService* request, the mapping between the two schemas is simply an XML transform. A service implementing the RPC operations defined above MAY choose whether to implement these natively (i.e. directly) or whether to transform them first using a transform service into the appropriate document based request.

6.5 Example

The purpose of the example here is to compare the XML message formats for two messages that make the same query upon the service. The first message is that generated by a SOAP-RPC client stub when the RPC operation *GridDataService::ExecuteQuery* is called. The second is the same query described in the more general *GridDataService* document format discussed in the previous sections.

The SOAP RPC message corresponding to the call

```
GridDataService::ExecuteQuery(
    "http://www.gridforum.org/dais/lang/SQL92",
    "select * from person where age > 21", 100)
```

is shown below (only the SOAP body is shown):

```
<SOAP-ENV:Body>
  <gds:executeQuery xmlns:gds="http://gridforum.org/dais/gds">
    <notation xsi:type="xsd:string">
      http://www.gridforum.org/dais/lang/SQL92
    </notation>
    <databaseName>Customer</databaseName>
    <query xsi:type="xsd:string">
      select * from person where age > 21
    </query>
    <maximum xsi:type="xsd:integer">100</maximum>
  </gds:executeQuery>
</SOAP-ENV:Body>
```

The corresponding *GridDataService* document message has two parts, the *gridDataServiceRequest* part that describes the query, and the *gridDataTransportDescription* that requests a full synchronous return (pull) of no more than 100 rows.

```
<gridDataServiceRequest>
  <executeStatement>
    <dbStatement notation="http://www.gridforum.org/dais/lang/SQL92"
      returnFormat=
        "http://gridforum.org/dais/schema/webRowSet.xsd"
      statementType="query"
      databaseName="Customer">
      <expression>select * from person where age > 21</expression>
    </dbStatement>
    <statementId>statement1</statementId>
  </executeStatement>

  <GridTransportDescription direction="get" mode="direct"
    maxSize="100" timeout="0">
    <statementId>statement1</statementId>
  </GridTransportDescription>
</gridDataServiceRequest>
```

Deleted: <http://www.gridforum.org/dais/lang/SQL92>

Deleted: operation

It can be seen that, assuming some default behaviour for the Remote Procedure Call, both these messages will result in the same query upon the database and are related by an XML transform.

7 Conclusions

Acknowledgements

This work is partly funded by the OGSA-DAI project, which includes support from Oracle UK, IBM and the UK e-Science Programme. Many interactions with members of the OGSA-DAI team have been important in shaping the ideas behind this document. Gavin McCance is employed on the EU DataGrid project.

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

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.

Deleted: <http://www.ibm.com/developerworks/library/ws-transpec/>,

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.

Deleted: <http://www.w3.org/TR/wsdl>

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.

Deleted: <http://www.cs.man.ac.uk/grid-db>

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

Deleted: <http://www.w3.org/TR/xmlschema-1/>,

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.

S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham and C. Kesselman, Grid Service Specification, Draft 3, <http://www.gridforum.org/ogsi-wg>, 2002.

Deleted: <http://www.gridforum.org/ogsi-wg>