GFD-I

Neil P Chue Hong, University of Edinburgh
Amy Krause, University of Edinburgh
Susan Malaika, IBM
Gavin McCance, University of Glasgow
Simon Laws, IBM
James Magowan, IBM
Norman W Paton, University of Manchester
Greg Riccardi, Florida State University

GGF Database Access and Integration Services Working Group
Category: INFORMATIONAL

16th February 2003

Grid Database Service Specification

Status of This Memo

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

Copyright Notice

Abstract

Data management systems are central to many applications across multiple domains, and play a significant role in many others. Web services provide implementation neutral facilities for describing, invoking and orchestrating collections of networked resources. The Open Grid Services Architecture (OGSA) extends Web Services with consistent interfaces for creating, managing and exchanging information among Grid Services, which are dynamic computational 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.

Contents

Neil P Chue Hong, University of Edinburgh
Amy Krause, University of Edinburgh
Susan Malaika, IBM
Gavin McCance, University of Glasgow
Simon Laws, IBM
James Magowan, IBM
Norman W Paton, University of Manchester
Greg Riccardi, Florida State University

GGF Database Access and Integration Services Working Group
Category: INFORMATIONAL
16th February 2003

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, Raman 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 information about itself to allow the service to be used given the specification of the service and the metadata provided by the service. Service metadata is made available using Service Data Elements [Tuecke 02].
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 specification does not seek to mandate this.
6. The specification for the most part follows the document approach to service description. As such, there are relatively few operations, and most functionality is described using document definitions. This has the standard advantages (and disadvantages) of the document-based approach. A Remote Procedure Call (RPC) style of interface is included in Section 6 to illustrate how it can be used as an alternative or complement to the document 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. The OGSA-DAI project (http://www.ogsa-dai.org) is developing a reference implementation and user documentation for the services described in this document. The European DataGrid

project is providing an implementation and user documentation for a Remote Procedure Call (RPC) based service (http://cern.ch/hep-proj-spitfire).

8. The specification seeks to support higher-level information-integration and federation services. As such, features such as service description and data delivery support, have been designed with a view to providing the necessary 'hooks' for the developers of such services.
9. This document is intended to specify an interface for Grid Database Services in a precise manner. A companion document, the Grid Database Service Primer [Chue Hong 03] should be read in conjunction with this specification, and may well be a better starting point than this document for anyone seeking an overview of the proposal.

Some familiarity 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 5 (4th November 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 (service requester) and a data source (service provider), 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. A Grid Database Service instance MAY map directly to a database session or connection of an underlying data management system, but this is not mandatory. For example, a common practice for large-scale systems is to share connections across many clients (called connection pooling). 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. The following operation is specified on *GridDataService*:
    - *GridDataService::perform*.

  This operation allows a request to be defined and for the execution of requests to be controlled. Typically execution control means executing or terminating a request. As the *perform* 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. A *GridDataService* MUST support the *GridDataService* portType.
- *GridDataTransport.* This is a new portType that provides functionality for communicating results between Grid Database Services. The following operations are specified on *GridDataTransport*:
    - *GridDataTransport::get*.
    - *GridDataTransport::put*.

  A Grid Database Service SHOULD support the *GridDataTransport* portType.

- *GridDataRPCService*. This is a new portType that provides a subset of the functionality provided by the *GridDataService* portType, but using operations following an RPC-based style. The operations supported are described in Section 6. A Grid Database Service SHOULD support the *GridDataRPCService* portType.
- *GridService*. This is the mandatory portType for Grid Services that provides access to information about a service [Tuecke 02], which is described as Service Data Elements (SDEs). The specification provides XML Schema definitions for SDEs for *GridDataService*s, and as such provides information on the content and capabilities of the service. A Grid Database Service MUST support the *GridService* portType.
- *NotificationSource*. This is the optional portType for Grid Services that enables a service to send notification messages [Tuecke 02]. The specification comments on how OGSA notifications can be used in the context of Grid Database Services.

The specification includes paradigm-dependent and paradigm-independent parts. 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 XML types used to convey parameters into and results from the operations of the *GridDataService* portType; and
- several of the SDEs of a service – for example, those relating to schemas.

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 of Grid Database Services directly. Thus it is assumed for now that a Grid Database Service Factory can be discovered using an existing service registry. This is illustrated in Figure 1, in which a requester uses the *FindServiceData* operation of a *GridServiceRegistry* to identify a *GridDataServiceFactory*. The *CreateService* operation on the factory is then used to create a *GridDataService* instance that might, for example, represent a session over a database. 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.

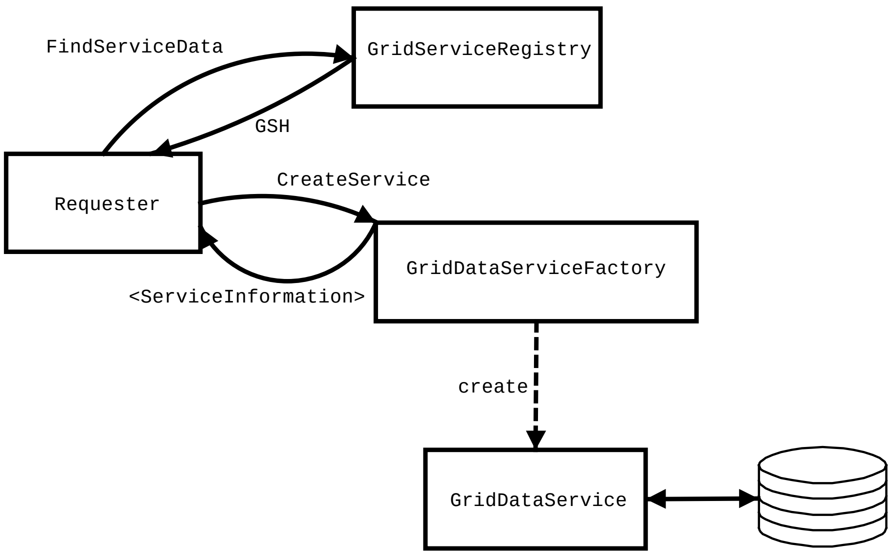

Figure 1. Creating a Grid Data Service.

Once a service instance has been created or discovered, there are in general three parts to an interaction of a requester with a *GridDataService,* as illustrated in Figure 2. 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 part, 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 to a third party. In the optional third part, several GridDataServices may exchange messages by way of the *GridDataTransport* portType.

In essence, this document provides a specification of the SDEs and portTypes of the *GridDataServiceFactory* illustrated in Figure 1 and the *GridDataService* illustrated in Figure 2.

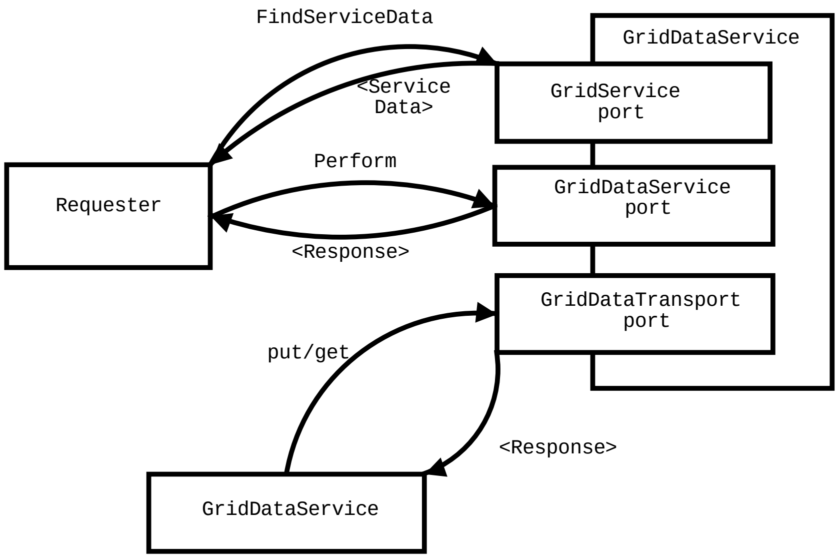

Figure 2. Requester Using Grid Data Service Ports.

The remainder of this document provides detailed descriptions of the portTypes and associated service data for Grid Database Services. In essence, each section describes a different portType, as follows: Section 3 indicates how the *Factory* portType of the Grid Service Specification is used to create Grid Database Services; Section 4 describes the *GridDataService* portType, including the document structure provided to the *perform* operation and the use of this portType with Relational and XML databases; Section 5 describes the *GridDataTransport* portType, which supports the delivery of data to and from Grid Database Services; Section 6 describes the *GridDataRPCService* portType, which provides an alternative to the document model supported by the *GridDataService* portType; Section 7 describes the *NotificationSource* portType; and Section 0 presents some conclusions.

3. Factory PortType

GridDataService instances are created by services supporting the *Factory* portType of the Grid Service Specification [Tuecke 02]. The *ServiceParameters* input to *Factory::CreateService* provides a document that is specific to the factory and the services it creates. The document passed to *Factory::CreateService* can, for example, contain configuration data for the *GridDataService* the factory will create.

A *GridDataServiceFactory* is configured to create *GridDataServices*. The factory is related to a number of data resources. The current specification covers the cases where the resource is a relational database or an XML database. A *GridDataService* is associated with a single data resource.

Factory metadata is exposed to publish configuration information and other qualities of the GDS instance that may be created. The factory metadata may contain information such as

- the location of the data resources,
- product information, and

- supported drivers and query/update languages.

When requesting the creation of a *GridDataService* the *ServiceParameters* argument specifies the data resource the GDS will be associated with as well as its capabilities. The mechanism of this specification is not specified here. It could be as simple as providing the name of a pre-defined data resource. Alternatively, more complex schemes can be imagined where the desired request is submitted to the factory and the factory creates a GDS instance best able to satisfy the request.

4. 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.

4.1 GridDataService PortType: Service Data Descriptions and Elements

4.1.1 Paradigm-Independent Service Data

This service data is present regardless of the type of grid resource with which the service is associated, and can be considered to be structural SDEs.

4.1.1.1 Request Handling

The *GridDataService* portType is associated with serviceData elements conformant to the following *serviceDataDescription* elements. The types that these service data elements refer to are defined in the appendix.

```
<gsdl:serviceDataDescription
      name="ActivityTypes"
      element="gdstypes:ActivityQNameType"
      minOccurs="1"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            A list of names of activity types that may appear within
            the Request element in the GridDataServicePerform document.
      </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
      name="DefinedRequests"
      element="gdstypes:RequestType"
      minOccurs="0"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            Available request documents.  Requests are added to the
            GridDataService by including them in the
            GridDataServicePerform document.
      </wsdl:documentation>
```

```
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
      name=”RunningRequests”
      element=”gdstypes:GridDataServiceResponseType”
      minOccurs=”0”
      maxOccurs=”unbounded”
      mutability=”mutable”>
      <wsdl:documentation>
            The status of any requests currently being executed.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

4.1.1.2 Data Resource

The data resource identifies the grid resource that the GDS operates on. Like a URI, it aggregates information relating to the physical location of the resource, local path information and the scheme that is used to access the resource. The data resource references other named SDEs that contain detailed information. In the factory, the data resource is used to identify valid combinations from a choice of detailed configuration information. In a *GridDataService* instance, the data resource simply defines the instances chosen configuration. The SDEs referenced by the data resource may be either paradigm-dependent or independent, for example, the path could equally refer to a relation database or XML:DB collection depending on the *Driver*, and *DataResourceManagementSystem* configuration.

```
<gsdl:serviceDataDescription
      name=”DataResource”
      element=”xsd:DataResourceType”
      minOccurs=”1”
      maxOccurs=”1”
      mutability=”mutable”>
      <wsdl:documentation>
            An XML document that contains references to SDEs describing
            the data resource to which this GridDataService has access.
            This includes, but is not limited to, the name of the data
            resource, the location of the data resource, the drivers
            that are supported.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

4.1.1.3 Driver

The driver SDE gives information about the type and version of the driver that can be used to access a data resource. The data resource references the names of the drivers it supports.

```
<xs:complexType name="DriverType">
      <xs:attribute name="name" type="xs:QName" use="required"/>
      <xs:attribute name="commonName" type="xs:string" use="required"/>
      <xs:attribute name="version" type="xs:string" use="optional"/>
</xs:complexType>

<gsdl:serviceDataDescription
      name="driver"
      element="tns:DriverType"
      minOccurs="1"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            A description of a driver that can be used to access a data
resource.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

An example for a XML database driver for the Xindice database:

```
<serviceData name=”driver”>
      <name> xmldb </name>
      <commonName>
            org.apache.xindice.client.xmldb.DatabaseImpl
      </commonName>
      <version> 20020101 </version>
</serviceData>
```

An example for a relational database driver for the MySQL database:

```
<serviceData name=”driver”>
      <name> MySQL </name>
      <commonName>
            com.mysql.jdbc.Driver
      </commonName>
      <version> 2003-01-23 - Version 3.0.5 Gamma</version>
</serviceData>
```

4.1.1.4 Language

The language SDE allows the data resource to provide information about the languages supported.

```
<gsdl:serviceDataDescription
      name="language"
      element="LanguageType"
      minOccurs="1"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            A description of a language that can be used to access a
data resource.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

Examples:

```
<serviceData name=”dai:language”>
      <name> XPath1.0 </name>
      <type> XPath </type>
      <version> 1.0 </version>
</serviceData>

<serviceData name=”dai:language”>
      <name> XUpdate1.0 </name>
      <type> XUpdate </type>
      <version> 1.0 </version>
</serviceData>
```

```
<serviceData name=”dai:language”>
      <name> SQL92 </name>
      <type> SQL </type>
      <version> 92 </version>
</serviceData>
```

4.1.1.5 Data Resource Management System

The *DataResourceManagementSystem* SDE publishes information about the data management system that underpins the GDS.

```
<gsdl:serviceDataDescription
      name="dataResourceManagementSystem"
      element="DataResourceManagementSystemType"
      minOccurs="1"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            A description of the data resource management system that
contains the data to be accessed or updated.
      </wsdl:documentation>
```

The following is an example for an XML database:

```
<serviceData name=”dataResourceManagementSystem”>
      <name>xind1</name>
      <productName> Xindice </productName>
      <productVersion> 1.1 </productVersion>
      <vendorName> Apache XML </vendorName>
      <location> localhost:4080 </location>
</serviceData>
```

The following is an example for a relational database:

```
<serviceData name=”dataResourceManagementSystem”>
      <name>MySQLSystem1</name>
      <productName> MySQL </productName>
      <productVersion> 3.20 </productVersion>
      <vendorName> MySQL </vendorName>
      <location> database.MySQLtest.org:3333 </location>
</serviceData>
```

4.1.2 Service Data for Relational Database Services

A *GridDataService* for relational data is associated with SDEs conformant to the following *serviceDataDescription* elements. These service data elements are specific to relational databases and should be considered to be dynamic service data elements.

```
<gsdl:serviceDataDescription
      name=”Database”
      element=”gdstypes:DatabaseType”
      minOccurs=”0”
      maxOccurs=”unbounded”
      mutability=”mutable”>
      <wsdl:documentation>
            This physically describes the database include sizes etc.
      </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
      name=”DatabaseSchema”
      element=”gdstypes:databaseSchemaType”
      minOccurs=”0”
      maxOccurs=”unbounded”
      mutability=”mutable”>
      <wsdl:documentation>
            This describes the database metadata such as columns,
            column types, keys etc
      </wsdl:documentation>
</gsdl:serviceDataDescription>

<gsdl:serviceDataDescription
      name=”LanguageCapabilities”
      element=”gdstypes:SQLServiceData”
      minOccurs=”0”
      maxOccurs=”1”
      mutability=”mutable”>
      <wsdl:documentation>
            This describes the dialect of SQL supported by the service.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

4.1.2.1 Database

This SDE provides some information on the physical properties of a database.

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

<xs:complexType name="databaseServiceData">
      <xs:element name="databaseSize" type="sizeDef"
                  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>
```

The above definitions are based on portions of the metadata supplied by JDBC.

4.1.2.2 Database Schema

The *DatabaseSchema* SDE for a Relational Database Service contains an XML document describing the tables, columns and data types supported in a database. The document SHOULD conform to the schema in Section 9.2. A service MAY also provide information about stored procedures and triggers. Some databases may choose to expose stored procedures as the only *Service Data*.

4.1.2.3 Language Capabilities

Although the *language* SDE provides coarse-grained information on the dialect of SQL supported by a relational GDS, versions of specific products vary in the capabilities provided, so a service can provide a more precise characterization of the language supported using the *LanguageCapabilities* SDE.

The schema for the SQL capability list in Section 9.3 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 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 in Section 9.3; a service SHOULD implement all of the schema given, and MAY implement further parts of the schema described in [ISO 9075].

4.1.3 Service Data for XML Database Services

These service data elements are specific to XML:DB databases and should be considered to be dynamic service data elements.

4.1.3.1 Collection Structure

The *Collections* 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 *CollectionSchema* SDE:

```
<xs:complexType name="collectionType">
  <xs:sequence>
       <xs:element  name="collection"
                    type="collectionType"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:QName" use="required"/>
</xs:complexType>

<gsdl:serviceDataDescription
      name="collectionStructure"
      element="collectionType"
      minOccurs="1"
      maxOccurs="1"
      mutability="mutable">
      <wsdl:documentation>
            The collection structure of an XML database.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

For example, given an XML database with the following collection structure:

- db
- db/mydocuments
- db/mytestdb
- db/mytestdb/moredocuments

the collection SDE is as follows:

```
<serviceData name="collectionStructure">
  <collection name="db">
    <collection name="db/mydocuments"/>
    <collection name="db/mytestdb">
      <collection name="db/mytestdb/moredocuments"/>
    </collection>
  </collection>
</serviceData>
```

4.1.3.2 CollectionSchema

A collection MAY have associated schemas that all documents in the collection must satisfy. These schemas can be exposed with the following *CollectionSchema* SDE:

```
<gsdl:serviceDataDescription
      name="collectionSchema"
      element="xsd:anyURI"
      minOccurs="1"
      maxOccurs="unbounded"
      mutability="mutable">
      <wsdl:documentation>
            The collection schemas of an XML database.
      </wsdl:documentation>
</gsdl:serviceDataDescription>
```

4.2 GridDataService PortType: Operations and Messages

GridDataService::perform
Perform a statement on a GridDataService.
Input:

- *gridDataServicePerform*: The document that describes the operation to be performed.

Output:

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

Fault:

- *gridDataServiceSystemException*: An exception report that the client can take no action to resolve
- *gridDataServiceUserException*: An exception report that the client can resolve
- *NameNotUniqueException*: All top level elements in the *gridDataServicePerform* document are named. If a name is used that is not unique, in the context of the GDS lifetime, an exception is raised
- *NameNotFoundException*: Top level elements reference other top level elements by name. If these references are not resolved an exception is raised.

Every *GridDataService* MUST implement the *GridDataService::perform* operation.

The behaviour of the *GridDataService*, following a call to *GridDataService::perform*, is characterised by the contents of the *GridDataServicePerform* document passed in as a parameter.

The top-level elements dictate what action should be taken:

- *Request* defines a request that should be stored by the GDS for future execution.
- *Execute* indicates that a stored request should be run. The name of the execute element identifies the running request for future reference.
- *Terminate* stops a named running request.

The *GridDataServicePerform* document is described in more detail in Section 4.3.1.1.

4.3 GridDataService PortType: Types

4.3.1 Paradigm Independent Types

4.3.1.1 GridDataService PortType: GridDataServicePerform

This document controls the definition, execution and termination of *Requests*. The following shows the structure of a *gridDataServicePerform* document. Three example activities are included, where each activity describes a component of the request being made to the service.

```
<GridDataServicePerform>
  <Request name="request1">
    <Parameter name="age"/>

    <SQLStatement name="statement1" ...>
      ...
    </SQLStatement>

    <Delivery name="d1">
      ...
    </Delivery>

    <!-- Extensibility Element – through substitution groups -->

  </Request>

  <Execute name=”execute1” request=”request1”>
    ...
  </Execute>

  <!-- Extensibility Element -->
  <!-- Other control elements. Only Terminate is currently defined -->

</GridDataServicePerform>
```

Execute initiates the execution of a named request. *Parameter* activities of the named request may be initialized through the *Execute* activity using *WithParameter*.

Terminate stops the running request.

```
<xsd:complexType name="ExecuteType">
      <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
```

```
                  <xsd:sequence>
                        <xsd:element name="WithParameter"
                                     type="tns:WithParameterType"
                                     minOccurs="0"
                                     maxOccurs="unbounded"/>
                  </xsd:sequence>
                  <xsd:attribute name="request" type="xsd:string"/>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TerminateType">
      <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                  <xsd:attribute name="runningRequest"
                                 type="xsd:string"/>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="GridDataServicePerformType">
      <xsd:sequence>
            <xsd:element name="Request"
                         type="tns:RequestType"
                         minOccurs="0"
                         maxOccurs="unbounded"/>
            <xsd:element name="Execute"
                         type="tns:ExecuteType"
                         minOccurs="0"
                         maxOccurs="unbounded"/>
            <xsd:element name="Terminate"
                         type="tns:TerminateType"
                         minOccurs="0"
                         maxOccurs="unbounded"/>
            <!-- extensibility element -->
      </xsd:sequence>
</xsd:complexType>
```

Request describes an action the GDS instance can perform against the data resource that it represents. It does this using a collection of linked *Activities*.

```
<xsd:complexType name="RequestType">
      <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                  <xsd:sequence>
                        <xsd:element ref="tns:Activity"
                                     minOccurs="0"
                                     maxOccurs="unbounded" />
                        <!-- extensibility via substitutionGroup -->
                  </xsd:sequence>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
```

Each included activity must define a *name* attribute. This name must be unique within the *gridDataServicePerform* document within which it is defined. This name associates returned results and status information with an activity.

```
<xsd:complexType name="ActivityType" abstract="false">
    <xsd:attribute name="name" type="xsd:ID" use="required"/>
</xsd:complexType>
```

Activities define inputs and outputs. The unique name of an output is referenced by the *from* attribute of an input. The effect is to link two activities together.

```
<xsd:complexType name="ActivityInputType"
                 abstract="true"
                 mixed="true">
    <xsd:sequence>
        <xsd:element name="UseParameter"
                     type="tns:UseParameterType"
                     minOccurs="0"
                     maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="from" type="xsd:IDREF" use="optional"/>
</xsd:complexType>

<xsd:complexType name="ActivityOutputType"
                 abstract="true"
                 mixed="true">
    <xsd:sequence>
        <xsd:element name="UseParameter"
                     type="tns:UseParameterType"
                     minOccurs="0"
                     maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:ID" use="required"/>
</xsd:complexType>
```

The above abstract types are realized as concrete activities in accordance with the configuration of a *GridDataService* instance. Concrete activities appear in *Request* by virtue of an *xsd:substitutionGroup* formed around *Activity*, for example,

```
<xsd:complexType name="ExampleActivityType">
   <xsd:complexContent>
       <xsd:extension base="tns:ActivityType">
          <xsd:sequence>
             <xsd:element name="ValueIn" maxOccurs="unbounded">
                <xsd:complexType mixed="true">
                   <xsd:complexContent>
                      <xsd:extension base="tns:ActivityInputType">
                      </xsd:extension>
                   </xsd:complexContent>
                </xsd:complexType>
             </xsd:element>
             <xsd:element name="ValueOut">
                <xsd:complexType mixed="true">
                   <xsd:complexContent>
                      <xsd:extension base="tns:ActivityOutputType">
                      </xsd:extension>
                   </xsd:complexContent>
                </xsd:complexType>
             </xsd:element>
          </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

 <xsd:element name="ExampleActivity"
              type="tns:ExampleActivityType"
              substitutionGroup="tns:Activity"/>
```

In support of the configurable and extensible nature of *Request*, concrete activity types are listed in the *ActivityTypes* SDE. The sub-elements that a particular activity defines are dependent on the implementation of that activity.

Each request will define one or more activities. The sequence of activity execution is defined by associating the input of one activity with the output of another activity. The *from* attribute of input specifies the unique name of an output.

A default activity, *Parameter*, allows an element value within an activity to be substituted at execution time.

4.3.1.2 GridDataService PortType: GridDataServiceResponse

The result of *perform* corresponds to the following XML Schema definition:

```
<xsd:complexType name="ResultType">
  <xsd:sequence>
    <xsd:any namespace="##any"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="ResponseType">
  <xsd:sequence>
    <xsd:element name="Result"
                 type="tns:ResultType"
                 minOccurs="0"
                 maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="GridDataServiceResponseType">
  <xsd:sequence>
    <xsd:element name="Response" type="tns:ResponseType"/>
  </xsd:sequence>
</xsd:complexType>
```

When a *gridDataServicePerfom* document executes a *Request*, each activity in *Request* is performed on demand, as defined by the activity associations. The results are collected to yield a *gridDataServiceResponse*. The result of each activity is represented as a *Result* element.

When an activity fails, no subsequent statement is performed.

4.3.1.3 Grid Data Delivery Type

The delivery of data, both to and from a GDS, is important to most requests directed to a GDS. As such, a GDS SHOULD support the *Delivery* activity, with a type defined as follows.

```
<xsd:complexType name = “DeliveryFromLocalType”>
  <xsd:annotation>
    <xsd:documentation>
        A local named endpoint.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tns:ActivityInputType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name = “DeliveryToLocalType”>
  <xsd:annotation>
    <xsd:documentation>
        A local named endpoint.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tns:ActivityOutputType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name = “DeliveryURIType”>
  <xsd:annotation>
    <xsd:documentation>
        A URI specifying an external endpoint using the format
        scheme://[user:pass@]host[:port]/location[?parameters].
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:anyURI"/>
</xsd:complexType>

<xsd:complexType name = “DeliveryGDTType”>
  <xsd:annotation>
    <xsd:documentation>
        A URI specifying the GSH and name of a GDT.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice
  <xsd:restriction base="xsd:anyURI"/>
</xsd:simpleType>

<xsd:complexType name = “DeliveryWaitType”>
  <xsd:annotation>
    <xsd:documentation>
        When a wait type is used delivery does nothing. It waits until
        the GDS is contacted by an external service requesting or
        supplying the data.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="DeliveryActivityType">
```

```
  <xsd:complexContent>
    <xsd:extension base="tns:ActivityType">
      <xsd:sequence>
        <xsd:choice minOccurs=”1”>
          <xsd:element name="FromLocal"
                       type="tns:DeliveryFromLocalType"/>
          <xsd:element name="FromURI"   type="tns:DeliveryURIType"/>
          <xsd:element name="FromGDT" type="tns:DeliveryGDTType"/>
          <xsd:element name="FromWait"   type="tns:DeliveryWaitType"/>
        </xsd:choice>
        <xsd:choice minOccurs=”1” maxOccurs=”unbounded”>
          <xsd:element name="ToLocal" type="tns:DeliveryToLocalType"/>
          <xsd:element name="ToURI"   type="tns:DeliveryURIType"/>
          <xsd:element name="ToGDT"   type="tns:DeliveryGDTType"/>
          <xsd:element name="ToWait"   type="tns:DeliveryWaitType"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Note that a restriction is placed to allow only one delivery source.

The corresponding *Deliver* element in the *GridDataServicePerform* is:

```
<xsd:element name="Delivery"
             type="DeliveryActivityType"
             substitutionGroup="tns:Activity"/>
```

4.3.2 Relational Database Service Types

Each call on the *GridDataService::perform* operation must provide a *gridDataServicePerform* document as input and will receive a *gridDataServiceResponse* document as its result. Just one language is specific to Relational Database Service Types: SQL.

There are four types of SQL requests and in all cases an indication of success or failure is returned:

1. SQLQueryStatement - SELECT: This is a SQL query and the result will be a WebRowSet
2. SQLUpdateStatement - INSERT, UPDATE or DELETE: No WebRowSet is returned
3. CALL to a stored procedure; optionally output parameters together with zero or more WebRowSets are returned
4. SQL function invocation: Either a scalar value or a WebRowSet is returned

Types 3 and 4 are currently not specified.

Exceptions and SQLError Codes should they occur should be passed back within the Faults returned from invoking GridDataService::perform

4.3.2.1 SQLQueryStatement

The SQLQueryStatement has the following XML schema:

```
<xsd:complexType name=”SQLParameterType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
    <xsd:attribute name=”Position” type=”xsd:int”/>
</xsd:complexType>

<xsd:complexType name=”ExpressionType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name=”ResultType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityOutputType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SQLQueryStatementType">
     <xsd:complexContent>
        <xsd:extension base="tns:ActivityType">
          <xsd:sequence>
            <xsd:element name=”SQLParameter”
                         type="tns:SQLParameterType"
                         minOccurs=”0”
                         maxOccurs=”unbounded”/>
            <xsd:element name="DataResource"
                         type="xsd:string"
                         minOccurs=”0”
                         maxOccurs=”1”/>
           <xsd:element name=”Expression” type=”tns:ExpressionType”/>
           <xsd:element name=”Result” type=”tns:ResultType”/>
          </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>
```

The response will be returned as XML conforming to the WebRowset XML. A WebRowSet defines an XML format for the result of a SQL query. WebRowSets are being defined in JSR 114 (www.jcp.org/jsr/detail/114.jsp) which is about to go into public review very soon, WebRowSets permit the update of query results and for the updates to take effect on the relevant tables. WebRowSet updates are outside the scope of this draft of the grid database service specification.

4.3.2.2 SQL UpdateStatement

The SQLUpdateStatement has the following XML schema:

```
<xsd:complexType name="SQLUpdateStatementType">
     <xsd:complexContent>
          <xsd:extension base="tns:ActivityType">
           <xsd:sequence>
            <xsd:element name=”SQLParameter”
                         type="tns:SQLParameterType"
                         minOccurs=”0”
                         maxOccurs=”unbounded”/>
            <xsd:element name="DataResource"
                         type="xsd:string"
                         minOccurs=”0”
```

```
                        maxOccurs=”1”/>
           <xsd:element name=”Expression” type=”tns:ExpressionType”/>
           <xsd:element name=”Result” type=”tns:ResultType”/>
          </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>
```

This returns a simple response indicating the number of items affected by the update.

4.3.3 XML Database Service Types

Each call on the *GridDataService::perform* operation must provide a *gridDataServicePerform* 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.

4.3.3.1 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.

The XPath query activity has the following XML schema:

```
<xsd:complexType name=”NamespaceType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
    <xsd:attribute name=”prefix” type=”xsd:string”/>
</xsd:complexType>

<xsd:complexType name=”CollectionType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name=”ResourceIdType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name=”ExpressionType” mixed="true">
    <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="XPathStatementType">
            <xsd:complexContent>
                  <xsd:extension base="tns:ActivityType">
                        <xsd:sequence>
```

```
                              <xsd:element name="dataresource"
type="xsd:string"/>
            <xsd:element name="namespace" type="tns:NamespaceType"
minOccurs=”0” maxOccurs=”unbounded”/>
                              <xsd:element name="collection"
type="tns:CollectionType"/>
                              <xsd:element name="resourceID"
type="tns:ResourceIDType" minOccurs=”0” maxOccurs=”unbounded”/>
                              <xsd:element name="expression"
type="tns:ExpressionType"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
</xsd:complexType>
```

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

```
<ogsadai:posting
      xmlns:dais=”http://www.gridforum.org/namespaces/2003/dais”>
      <to>Bob</to>
      <from>Alice</from>
      <subject>Example</subject>
      <body>This is an example for a document</body>
</ogsadai:posting>
```

The following is a document for a single query request using the XPath query language. Parameters and namespace definitions are used.

```
<Parameter name=”thename”/>

<XpathStatement name=”xpath1”>
      <namespace prefix=”ogsadai”>
            “http://www.gridforum.org/namespaces/2003/dais”
      </namespace>
      <resourceID>doc1</resourceID>
      <expression>/ogsadai:posting/to[text()=”<UseParameter
name=”thename”>”]</expression>
<XPathStatement>
```

4.3.3.2 XUpdate

The XUpdate language is supported by the XML:DB API. It allows for inserting, deleting or modifying nodes of an XML document. The working draft is dated 14th September 2001.

For an XUpdate request, we define the *XUpdateStatement* activity which has the following XML schema:

```
<xsd:complexType name="XUpdateStatementType">
            <xsd:complexContent>
                  <xsd:extension base="tns:ActivityType">
                        <xsd:sequence>
                              <xsd:element name="dataResource"
type="xsd:string"/>
            <xsd:element name="namespace" type="tns:NamespaceType"
minOccurs=”0” maxOccurs=”unbounded”/>
            <xsd:element name="collection" type="tns:CollectionType"/>
            <xsd:element name="resourceID" type="tns:ResourceIDType"
minOccurs=”0” maxOccurs=”unbounded”/>
            <xsd:element name="expression" type="tns:ExpressionType"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
</xsd:complexType>
```

The following is an example update:

```
<XupdateStatement name=”Xupdate1”>
      <collection> /db/mail </collection>
      <expression>
         <xupdate:modifications version="1.0"
            xmlns:xupdate="http://www.xmldb.org/xupdate">
            <xupdate:insert-after select="/addresses/address[1]" >
                  <xupdate:element name="posting">
                        <To>Charlie</To>
                        <From>Alice</From>
                        <Body>Another example text.</Body>
                  </xupdate:element>
            </xupdate:insert-after>
         </xupdate:modifications>
      </expression>
</XUpdateStatement>
```

4.3.3.3 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.

```
<xsd:complexType name="XQueryStatementType">
            <xsd:complexContent>
                  <xsd:extension base="tns:ActivityType">
                        <xsd:sequence>
                              <xsd:element name="dataResource"
type="xsd:string"/>
            <xsd:element name="namespace" type="tns:NamespaceType"
minOccurs=”0” maxOccurs=”unbounded”/>
            <xsd:element name="collection" type="tns:CollectionType"/>
            <xsd:element name="resourceID" type="tns:ResourceIDType"
minOccurs=”0” maxOccurs=”unbounded”/>
            <xsd:element name="expression" type="tns:ExpressionType"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
</xsd:complexType>
```

4.3.3.4 Creating and Deleting Documents

The creation and deletion of documents is supported by the XML:DB API but is currently not supported by any XML language. To create a document, the following XML statement is sent in the body of the GDS request. This assumes that the document is loaded to the server as specified in a delivery description.

```
<xsd:complexType name="XMLResourceManagement">
     <xsd:complexContent>
          <xsd:extension base="tns:ActivityType">
          <xsd:sequence>

             <xsd:element name="collection" type="tns:CollectionType"/>
             <xsd:element name="expression" type="tns:requestType"/>
          </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>
```

A expression has the following XML schema:

```
<xsd:complexType name="requestType">
     <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
          <xsd:choice>
             <xsd:element name="createResource">
                <xsd:complexType>
                   <xsd:attribute name="name" type="xsd:string"
use="required"/>
                </xsd:complexType>
             </xsd:element>
             <xsd:element name="removeResource"/>
                  <xsd:complexType>
                  <xsd:attribute name="name" type="xsd:string"
use="required"/>
                </xsd:complexType>
          </xsd:choice>
      </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

For example, the removal of a resource named “res1” residing in the collection “/db/mail” of the data resource “dr1” is requested as follows:

```
<XMLCollectionManagementService name=”Management1”>
      <collection> /db/mail </collection>
      <expression>
            <removeResource name=”res1”/>
      </expression>
</XMLCollectionManagementService>
```

4.3.3.5 Creating and Deleting Collections

Creating and deleting of collections is supported by the XML:DB API but with very limited functionality. As there no standards to refer to at this point we propose the following XML schema:

```
<xsd:complexType name="XMLCollectionManagement">
     <xsd:complexContent>
         <xsd:extension base="tns:ActivityType">
```

```
          <xsd:sequence>
            <xsd:element name="collection" type="tnd:CollectionType"/>
            <xsd:element name="request" type="tns:requestType"/>
          </xsd:sequence>
        </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>
```

A expression has the following XML schema:

```
<xsd:complexType name="requestType">
     <xsd:complexContent>
        <xsd:extension base="tns:ActivityInputType">
          <xsd:choice>
            <xsd:element name="createCollection">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string"
use="required"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="removeCollection">
              <xsd:complexType>
                <xsd:attribute name="name" type="xsd:string"
use="required"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>
```

5. GridDataTransport PortType

The *GridDataTransport* portType provides additional operations required to support communication between *GridDataService* instances using the *toWait* and *fromWait* elements of *Delivery* activities, as introduced in Section 4.3.1.3. These operations are invoked implicitly during the execution of *GridDataService* requests – service requesters MUST NOT invoke these operations directly.

A *GridDataService* SHOULD support the *GridDataTransport* portType.

5.1.1 GridDataTransport PortType: Service Data Descriptions and Elements

The *GridDataTransport* portType is associated with SDEs conforming 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>
```

5.2 GridDataTransport PortType: Operations and Messages

The *GridDataTransport* portType has two operations.

GridDataService::get
Obtain data from the given activity in the given request, where that activity contains a *toWait* element.
Input:
- *RequestName:* The *name* of the *Request* to which data is being supplied.
- *ActivityName*: The *name* of the *Delivery* activity within the given *Request*.

Output:
- *Data*: The data to be delivered.

Fault:
- *GridDatabaseTransportFault*: The document that provides the fault.

Every *GridDataTransportService* MUST implement the *GridDataTransportService::get* operation.

GridDataService::put
Supply data to the given activity in the given request, where that activity contains a *fromWait* element.
Input:
- *RequestName:* The *name* of the *Request* to which data is being supplied.
- *ActivityName*: The *name* of the *Delivery* activity within the given *Request*.
- *Data*: The data to be delivered.

Output:
- *GridDataTransportResponse*: The document that reports on the outcome of the delivery.

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::put* operation.

6. GridDataRPCService PortType

Although the primary interface uses the web services document model, there are a number 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 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.

For this purpose, another port type is defined, the *GridDataRPCService*. A *GridDataRPCService* is divided into generic, relational specific and XML specific operations. The port type MUST implement all the generic operations, and MUST implement either the relational or the XML operations, depending on the database type.

The RPC operations defined here are synchronous, i.e. the direct transport is used; the RPC message is posted directly to the service and the result is returned to the client directly. Transactional support and session support are not discussed here. It is assumed that one GridDataRPCService is bound to one data source by its factory.

6.1 GridDataRPCService PortType: Service Data Descriptions and Elements

The *GridDataRPCService* portType MAY support any of SDEs of the *GridDataService* portType from Section 4.1.

6.2 GridDataRPCService PortType: Generic Operations and Messages

GridDataRPCService::PerformUpdate
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.
- *UpdateRequest*: The update string describing the request.

Output

- *UpdateResult*: The result of the update, indicating how many rows were affected.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The request could not be parsed.
- *InvalidOperation*: The request failed for some reason (e.g. a table that was referenced does not exist).

GridDataRPCService::PerformSchemaUpdate
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.
- *SchemaUpdateRequest*: The schema update string describing the request.

Output

- *SchemaUpdateResult*: The result of the schema update, indicating how many schema items were affected.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The request could not be parsed.
- *InvalidOperation*: The request failed for some reason (e.g. the referenced table does not exist). The response of the database should be encoded in this message, for example, the SQLSTATE and SQLCODE for SQL database errors.

GridDataRPCService::PerformQuery
Execute a database query using the specified query notation.
Input

- *Notation*: The query notation for the request. A database may support many query notations.
- *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.

Fault

- *InvalidNotation*: Requested notation is not supported.
- *InvalidFormat*: The query could not be parsed.
- *InvalidOperation*: The query failed for some reason (e.g. the referenced table does not exist).

6.3 GridDataRPCService PortType: Relational Model Specific Operations and Messages

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

GridDataRPCService::GetTableNames
Return the names of the tables held in the database.
Input

- *TableMatchList*: A wildcard to limit the number of tables returned. If omitted, all tables in the database are returned.

Output

- *TableList*: The list of tables in the database.

Fault

- *none.*

GridDataRPCService::GetTableDefinitions
Return the column names and types defined for a specific table.
Input

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

Output

- *TableDefinition*: The list of columns and their types in the database.

Fault

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

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

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

Output

- *TableSchema*: The full schema describing the table.

Fault

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

GridDataRPCService::PerformBulkRelationalLoad
Inserts the *BulkData* into the specified table. This operation is appropriate for small loads that can be reasonably accomplished in one transaction.
Input

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

Output

- *RowsInserted*: The number of rows successfully inserted.

Fault

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

6.4 GridDataRPCService PortType: XML Specific Operations and Messages

GridDataRPCService::GetSchema
Will return the XML schema used in the collection.
Input
- *none.*

Output
- *SchemaList*: The list of XML schemas of the collection.

Fault
- *none.*

GridDataRPCService::GetDocumentList
Returns the names of the documents held in the collection.
Input
- *none.*

Output
- *DocumentList*: The list of documents in the collection.

Fault
- none.

GridDataRPCService::PerformBulkXMLLoad
Inserts an XML document into the collection. This operation is appropriate for small loads that can be reasonably accomplished in one transaction.
Input
- *Document*: The document to insert.

Output
- *None*.

Fault
- *InvalidDocumentFormat*: The document is not formatted correctly according to its schema.
- *SchemaMismatch*: The document is not of the appropriate schema for the collection.

7. NotificationSource PortType

A *GridDataService* MAY allow notification of changes to its Service Data Elements using *subscribeByServiceDataName* [Tuecke 02].

8. Conclusions

This document has described a proposal for a collection of Grid Database Services, which includes support for multiple database paradigms and flexible data transport. The services proposed are Grid services, in that they conform to and make use of the Open Grid Services Architecture [Tuecke 02]. The intention is that the proposal be discussed in the context of the DAIS (www.cs.man.ac.uk/grid-db) Working Group of the Global Grid Forum (www.gridforum.org), with a view to a Proposed Recommendation document being produced in time for GGF7.

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.

9. Appendices

9.1 WSDL for the GridDataService PortType

```
<wsdl:definitions name="GridDataService"

targetNamespace="http://www.gridforum.org/namespaces/2003/gds"
                 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:gdstypes="http://www.gridforum.org/namespaces/2003/gds"

xmlns:gdsfaults="http://www.gridforum.org/namespaces/2003/gds"

xmlns:gsdl="http://www.gridforum.org/namespaces/2002/10/gridServices"

xmlns:tns="http://www.gridforum.org/namespaces/2003/gds"
                 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
  <xsd:schema
targetNamespace="http://www.gridforum.org/namespaces/2003/gds"

      xmlns:tns="http://www.gridforum.org/namespaces/2003/gds"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  elementFormDefault="qualified"
                  attributeFormDefault="unqualified">

      <xsd:import
namespace="http://www.gridforum.org/namespaces/2002/10/gridServices"
schemaLocation="core/types/service_data.xsd" />

      <!-- request parameters -->
      <xsd:complexType name="WithParameterType">
            <xsd:attribute name="name" type="xsd:IDREF"
use="required"/>
      </xsd:complexType>

      <xsd:complexType name="UseParameterType">
            <xsd:attribute name="name" type="xsd:IDREF"
use="required"/>
      </xsd:complexType>

      <!-- Base Activity types -->
    <xsd:complexType name="ActivityType" abstract="false">
            <xsd:attribute name="name" type="xsd:ID" use="required"/>
      </xsd:complexType>

      <xsd:complexType name="ActivityInputType" abstract="true"
mixed="true">
            <xsd:sequence>
                  <xsd:element name="UseParameter"
type="tns:UseParameterType" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="from" type="xsd:IDREF"
use="optional"/>
      </xsd:complexType>

      <xsd:complexType name="ActivityOutputType" abstract="true"
mixed="true">
            <xsd:sequence>
                  <xsd:element name="UseParameter"
type="tns:UseParameterType" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:ID" use="required"/>
      </xsd:complexType>

    <xsd:element name="Activity" type="tns:ActivityType"/>

      <!-- Request types -->
    <xsd:complexType name="RequestType">
           <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                        <xsd:sequence>
                              <xsd:element ref="tns:Activity"
minOccurs="0" maxOccurs="unbounded" />
                              <!-- extensibility through
substituionGroup -->
                        </xsd:sequence>
            </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

      <xsd:complexType name="ExecuteType">
           <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                        <xsd:sequence>
                              <xsd:element name="WithParameter"
type="tns:WithParameterType" minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                  <xsd:attribute name="request" type="xsd:string"/>
            </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

      <xsd:complexType name="TerminateType">
        <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                  <xsd:attribute name="runningRequest"
type="xsd:string"/>
            </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

      <xsd:complexType name="GridDataServicePerformType">
            <xsd:sequence>
                  <xsd:element name="Request" type="tns:RequestType"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="Execute" type="tns:ExecuteType"
minOccurs="0" maxOccurs="unbounded"/>
```

```
                  <xsd:element name="Terminate"
type="tns:TerminateType" minOccurs="0" maxOccurs="unbounded"/>
                  <!-- extensibility element -->
            </xsd:sequence>
      </xsd:complexType>

      <xsd:element name="GridDataServicePerform"
type="tns:GridDataServicePerformType"/>

      <!-- response types -->
      <xsd:complexType name="ResultType">
          <xsd:sequence>
            <xsd:any namespace="##any"/>
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:string"/>
      </xsd:complexType>

      <xsd:complexType name="ResponseType" mixed="true">
            <xsd:sequence>
                  <xsd:element name="Result" type="tns:ResultType"
minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string"/>
      </xsd:complexType>

      <xsd:complexType name="GridDataServiceResponseType">
            <xsd:sequence>
                  <xsd:element name="Response" type="tns:ResponseType"
minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:element name="GridDataServiceResponse"
type="tns:GridDataServiceResponseType"/>

      <!-- SDE types -->
      <xsd:complexType name="ActivityQNameType">
      <xsd:sequence>
                  <xsd:element name="ActivityQName" type="xsd:QName"/>
      </xsd:sequence>
      </xsd:complexType>

      <xsd:simpleType name="SDENameType">
            <xsd:restriction base="xsd:string">
            </xsd:restriction>
      </xsd:simpleType>

      <xsd:complexType name="SchemeType">
          <xsd:sequence>
            <xsd:element name="Language" type="tns:SDENameType"/>
            <xsd:element name="Driver" type="tns:SDENameType"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="LocationType">
          <xsd:sequence>
```

```
            <xsd:element name="DataResourceManagementSystem"
type="tns:SDENameType"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="PathType">
          <xsd:sequence>
            <xsd:element name="PathReference" type="tns:SDENameType"/>

            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="DataResourceType">
          <xsd:sequence>
                  <xsd:element name="Name" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="Scheme" type="tns:SchemeType"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="Location" type="tns:LocationType"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="Path" type="tns:PathType"
minOccurs="1" maxOccurs="1"/>
          </xsd:sequence>
      </xsd:complexType>

      <!-- default activity types -->
    <xsd:complexType name="ParameterType">
      <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
            </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

      <xsd:element name="Parameter" type="tns:ParameterType"
substitutionGroup="tns:Activity"/>

      <!-- Example Activity -->
      <xsd:complexType name="ExampleActivityType">
      <xsd:complexContent>
            <xsd:extension base="tns:ActivityType">
                  <xsd:sequence>
                        <xsd:element name="ValueIn"
maxOccurs="unbounded">
                              <xsd:complexType mixed="true">
                                    <xsd:complexContent>
                                          <xsd:extension
base="tns:ActivityInputType">
                                          </xsd:extension>
                                    </xsd:complexContent>
                              </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="ValueOut">
                              <xsd:complexType mixed="true">
                                    <xsd:complexContent>
                                          <xsd:extension
base="tns:ActivityOutputType">
                                          </xsd:extension>
                                    </xsd:complexContent>
                              </xsd:complexType>
                        </xsd:element>
                  </xsd:sequence>
            </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="ExampleActivity" type="tns:ExampleActivityType"
substitutionGroup="tns:Activity"/>

  </xsd:schema>
</wsdl:types>

  <wsdl:message name="performInputMessage">
    <wsdl:part name="parameters"
element="gdstypes:GridDataServicePerform"/>
  </wsdl:message>
  <wsdl:message name="performOutputMessage">
    <wsdl:part name="parameters"
element="gdstypes:GridDataServiceResponse"/>
  </wsdl:message>

  <wsdl:message name="GridDataServiceSystemException">
    <wsdl:part name="fault"
type="gdsfaults:GridDataServiceSystemExceptionType"/>
  </wsdl:message>
  <wsdl:message name="GridDataServiceUserException">
    <wsdl:part name="fault"
type="gdsfaults:GridDataServiceUserExceptionType"/>
  </wsdl:message>
  <wsdl:message name="NameNotFoundException">
    <wsdl:part name="fault"
type="gdsfaults:NameNotFoundExceptionType"/>
  </wsdl:message>
  <wsdl:message name="NameNotUniqueException">
    <wsdl:part name="fault"
type="gdsfaults:NameNotUniqueExceptionType"/>
  </wsdl:message>

  <wsdl:portType name="GridDataServicePortType">

    <wsdl:operation name="perform">
      <wsdl:input message="tns:performInputMessage"/>
      <wsdl:output message="tns:performOutputMessage"/>
      <wsdl:fault name="GridDataServiceUserException"
message="tns:GridDataServiceUserException"/>
      <wsdl:fault name="GridDataServiceSystemException"
message="tns:GridDataServiceSystemException"/>
      <wsdl:fault name="NameNotFoundException"
message="tns:NameNotFoundException"/>
      <wsdl:fault name="NameNotUniqueException"
message="tns:NameNotUniqueException"/>
    </wsdl:operation>
  </wsdl:portType>

  <gsdl:serviceDataDescription
```

```
            name="DataResource"
            element="xsd:anyType"
            minOccurs="1"
            maxOccurs="1"
            mutability="mutable">
            <wsdl:documentation>
               An XML document that contains information describing the
data resource to which this GridDataService has access. This includes,
but is not limited to, the name of the data resource, the location of
the data resource, the drivers that are supported etc.
            </wsdl:documentation>
  </gsdl:serviceDataDescription>

  <!-- other sections to contribute meta-data -->

  <gsdl:serviceDataDescription
            name="ActivityTypes"
            type="gdstypes:ActivityQNameType"
            minOccurs="1"
            maxOccurs="unbounded"
            mutability="mutable">
            <wsdl:documentation>
              A list of names of activity types that may appear in the
GridDataServiceRequest.
            </wsdl:documentation>
  </gsdl:serviceDataDescription>

  <gsdl:serviceDataDescription
            name="DefinedRequests"
            type="gdstypes:RequestType"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability="mutable">
            <wsdl:documentation>
               Available request definitions
            </wsdl:documentation>
   </gsdl:serviceDataDescription>

      <gsdl:serviceDataDescription
            name="RunningRequests"
            type="gdstypes:GridDataServiceResponseType"
            minOccurs="0"
            maxOccurs="unbounded"
            mutability="mutable">
            <wsdl:documentation>
             The status of any requests currently being executed
            </wsdl:documentation>
      </gsdl:serviceDataDescription>

      <gsdl:serviceData name="ActivityType">

      <gdstypes:ActivityType>gdstypes:Parameter</gdstypes:ActivityType>
      </gsdl:serviceData>

</wsdl:definitions>
```

9.2 XML Schema for Relational Database Schema

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

    <xs:complexType name="databaseSchematype">
        <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: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>
```

9.3 XML Schema for Relational Language Capabilities

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

```
<xs:complexType name="SQLServiceData">
      <xs:element name="SQLSupport" type="SQLSupportDef"
                        minOccurs="1" maxOccurs="1"/>
</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>
```

9.4 WSDL for GridDataRPCService PortType

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="GridRPCService"
    targetNamespace="http://www.gridforum.org/dais/GridRPCService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.gridforum.org/dais/GridRPCService.wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd1="http://www.gridforum.org/dais/GridRPCService.xsd1"
    xmlns:xsd2="http://www.gridforum.org/dais/GridRPCService.xsd2">

    <types>
        <xsd:schema
targetNamespace="http://www.gridforum.org/dais/GridRPCService.xsd1"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.gridforum.org/dais/GridRPCService.wsdl"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsd1="http://www.gridforum.org/dais/GridRPCService.xsd1">
</xsd:schema>
    </types>
    <message name="PerformBulkXMLLoadRequest">
        <part name="Document" type="xsd:anyType"/>
    </message>

    <message name="PerformUpdateRequest">
        <part name="Notation" type="xsd:string"/>
        <part name="UpdateRequest" type="xsd:string"/>
    </message>
    <message name="InvalidBulkDataFormatError">   </message>

    <message name="GetSchemaRequest">
        <documentation>This defines no parameters.</documentation>
    </message>

    <message name="GetFullSchemaResponse">
        <part name="TableSchema" type="xsd:string"/>
    </message>

    <message name="PerformSchemaUpdateRequest">
        <part name="Notation" type="xsd:string"/>
        <part name="SchemaUpdate" type="xsd:string"/>
    </message>

    <message name="OperationError">   </message>

    <message name="GetSchemaResponse">
        <part name="SchemaList" type="xsd:string"/>
    </message>

    <message name="NotationtError">   </message>

    <message name="InvalidDocumentFormatError">   </message>

    <message name="PerformBulkRelationalLoadResponse">
        <part name="RowsInserted" type="xsd:positiveInteger"/>
    </message>

    <message name="GetTableNamesResponse">
        <part name="TableList" type="xsd:string"/>
    </message>

    <message name="GetTableDefinitionsResponse">
        <part name="TableDefinition" type="xsd:string"/>
    </message>

    <message name="PerformBulkRelationalLoadRequest">
        <part name="TableName" type="xsd:string"/>
        <part name="BulkData" type="xsd:string"/>
    </message>

    <message name="SchemaMismatchError">   </message>

    <message name="GetFullSchemaRequest">
        <part name="TableName" type="xsd:string"/>
    </message>

    <message name="PerformQueryResponse">
        <part name="QueryResult" type="xsd:string"/>
    </message>

    <message name="PerformQueryRequest">
        <part name="Notation" type="xsd:string"/>
        <part name="Query" type="xsd:string"/>
        <part name="Maximum" type="xsd:positiveInteger"/>
    </message>

    <message name="GetTableDefinitionsRequest">
        <part name="TableName" type="xsd:string"/>
    </message>

    <message name="NonExistantTableError">   </message>

    <message name="GetDocumentListRequest">
        <documentation>This defines no parameters.</documentation>
    </message>

    <message name="PerformUpdateResult">
        <part name="UpdateResult" type="xsd:integer"/>
    </message>

    <message name="GetTableNamesRequest">
        <part name="TableMatchList" type="xsd:string"/>

    </message>

    <message name="PerformSchemaUpdateResponse">
        <part name="SchemaUpdateResult" type="xsd:integer"/>
    </message>

    <message name="PerformBulkXMLLoadResponse">   </message>

    <message name="GetDocumentListResponse">
        <part name="DocumentList" type="xsd:string"/>
    </message>

    <message name="FormatError">   </message>

    <portType name="GridRPCServicePortType">
        <operation name="PerformUpdate">
            <documentation>Execute a database update using the specified
query notation.</documentation>
            <input message="tns:PerformUpdateRequest"/>
            <output message="tns:PerformUpdateResult"/>
            <fault message="tns:NotationtError" name="InvalidNotation"/>
            <fault message="tns:OperationError"
name="InvalidOperation"/>
            <fault message="tns:FormatError" name="InvalidFormat"/>
        </operation>

        <operation name="PerformSchemaUpdate">
            <documentation>Execute a database schema update in the
specified query notation.</documentation>
            <input message="tns:PerformSchemaUpdateRequest"/>
            <output message="tns:PerformSchemaUpdateResponse"/>
            <fault message="tns:NotationtError" name="InvalidNotation"/>
            <fault message="tns:OperationError"
name="InvalidOperation"/>
            <fault message="tns:FormatError" name="InvalidFormat"/>
        </operation>

        <operation name="PerformQuery">
```

```
            <documentation>Will execute a database query using the
specified query notation.</documentation>
            <input message="tns:PerformQueryRequest"/>
            <output message="tns:PerformQueryResponse"/>
            <fault message="tns:NotationtError" name="InvalidNotation"/>
            <fault message="tns:OperationError"
name="InvalidOperation"/>
            <fault message="tns:FormatError" name="InvalidFormat"/>
        </operation>

        <operation name="GetTableNames">
            <documentation>Return the names of the tables held in the
database.</documentation>
            <input message="tns:GetTableNamesRequest"/>
            <output message="tns:GetTableNamesResponse"/>
        </operation>

        <operation name="GetTableDefinitions">
            <documentation>Return the column names and types defined for
a specific table.</documentation>
            <input message="tns:GetTableDefinitionsRequest"/>
            <output message="tns:GetTableDefinitionsResponse"/>
            <fault message="tns:NonExistantTableError"
name="NonExistantTable"/>
        </operation>

        <operation name="GetFullSchema">
            <documentation>Returns the full schema of the table or view
requested, including all the stored triggers and
indices.</documentation>
            <input message="tns:GetFullSchemaRequest"/>
            <output message="tns:GetFullSchemaResponse"/>
            <fault message="tns:NonExistantTableError"
name="NonExistantTable"/>
        </operation>

        <operation name="GetSchema">
            <documentation>Will return the XML schema used in the
collection.</documentation>
            <input message="tns:GetSchemaRequest"/>
            <output message="tns:GetSchemaResponse"/>
            <fault message="tns:FormatError" name="fauldsd"/>
        </operation>

        <operation name="GetDocumentList">
            <documentation>Returns the names of the documents held in
the collection.</documentation>
            <input message="tns:GetDocumentListRequest"/>
            <output message="tns:GetDocumentListResponse"/>
        </operation>

        <operation name="PerformBulkRelationalLoad">
            <documentation>Inserts the bulkData into the specified
table. This operation is appropriate for small loads that can be
reasonably accomplished in one transaction.</documentation>
            <input message="tns:PerformBulkRelationalLoadRequest"/>
            <output message="tns:PerformBulkRelationalLoadResponse"/>
            <fault message="tns:SchemaMismatchError"
name="SchemaMismatch"/>
            <fault message="tns:InvalidBulkDataFormatError"
name="InvalidBulkDataFormat"/>
            <fault message="tns:NonExistantTableError"
name="NonExistantTable"/>
        </operation>

        <operation name="PerformBulkXMLLoad">
            <documentation>Inserts an XML document into the collection.
This operation is appropriate for small loads that can be reasonably
accomplished in one transaction</documentation>
            <input message="tns:PerformBulkXMLLoadRequest"/>
            <output message="tns:PerformBulkXMLLoadResponse"/>
            <fault message="tns:SchemaMismatchError"
name="SchemaMismatch"/>
            <fault message="tns:InvalidDocumentFormatError"
name="InvalidDocumentFormat"/>
        </operation>
    </portType>
</definitions>
```

Author Information

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

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

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

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

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

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

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

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

Trademarks

Intellectual Property Statement

Full Copyright Notice

References

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

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

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.

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

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

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

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

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

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

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

S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham and C. Kesselman, Grid Service Specification, Draft 5 http://www.gridforum.org/ogsi-wg, November 2002.