

The Design and Implementation of Grid Database Services in OGSA-DAI

Mario Antonioletti¹, Malcolm Atkinson², Rob Baxter¹, Andrew Borley³, Neil P Chue Hong¹, Brian Collins³, Neil Hardman³, Ally Hume¹, Alan Knox³, Mike Jackson¹, Amy Krause¹, Simon Laws³, James Magowan³, Norman W Paton⁴, Dave Pearson⁵, Tom Sugden¹, Paul Watson⁶ and Martin Westhead¹

¹EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

²National e-Science Centre, 15 South College Street, Edinburgh EH8 9AA, UK.

³IBM United Kingdom Ltd, Hursley Park, Winchester S021 2JN, UK.

⁴Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK.

⁵Oracle UK Ltd, Thames Valley Park, Reading RG6 1RA, UK.

⁶School of Computing Science, University of Newcastle, Newcastle-upon-Tyne NE1 7RU, UK.

Abstract

Initially, Grid technologies were principally associated with supercomputer centres and large-scale scientific applications in physics and astronomy. They are now increasingly seen as being relevant to many areas of e-Science and e-Business. The emergence of the Open Grid Services Architecture (OGSA), to complement the ongoing activity on web services standards, promises to provide a service-based platform that can meet the needs of both business and scientific applications. Early Grid applications focused principally on the storage, replication and movement of file-based data. Now the need for the full integration of database technologies with Grid middleware is widely recognised. Not only do many Grid applications already use databases for managing metadata, but increasingly many are associated with large databases of domain-specific information (e.g. biological or astronomical data).

This paper describes the design and implementation of OGSA-DAI, a service-based architecture for database access over the Grid. The approach involves the design of Grid Data Services that allow consumers to discover the properties of structured data stores and to access their contents. The initial focus has been on support for access to Relational and XML data, but the overall architecture has been designed to be extensible to accommodate different storage paradigms. The paper describes and motivates the design decisions that have been taken, and illustrates how the approach supports a range of application scenarios. The OGSA-DAI software is freely available from www.ogsadai.org.uk.

1 Introduction

This paper examines how databases can be being integrated into OGSA Grid environments through the use of the OGSA-DAI (OGSA Data Access and Integration) service based architecture.

1.1 The Need For Databases In Grid Environments

Early Grid applications, mainly motivated by an attempt to cater to scientific-based computing, were often closely associated with devices or tools that read and/or generated flat files. Consequently, support for files rather than for the management of structured data had the highest profile in the early Grid toolkits (e.g., [1]). However, over time, the file management systems and registries associated with Grid toolkits themselves became complex, and database management systems were increasingly used to store Grid metadata (e.g., [2]). Contemporaneously, the requirements of the scientific computing community have become more sophisticated with, for example, biological and astronomical communities generating large quantities of data that increasingly use databases for storage and retrieval. Similarly, engineering, medical research, healthcare and many governmental systems can also take advantage of Grids that access and integrate multiple and distributed collections of structured data. This data needs to be made available and accessible to distributed sets of users and their applications, which makes these communities ideal candidates to adopt Grid-based infrastructures.

1.2 The OGSA Grid Environment

Web Services emerged from the business computing world with the aim of supporting business-to-business relationships in a language-neutral, platform independent way. They are loosely-coupled, distributed components that interact with each other through the exchange of messages. To achieve the goal of interoperability, a collection of specifications are currently progressing through standardisation bodies such as the W3C and OASIS. The architectural principles on which Web Services are based may be found in [3] and a recent overview is provided in [4].

Starting in 2001, researchers, initially led by Globus (<http://www.globus.org>) and IBM, began developing new Grid standards and technology. The aim was to merge the understanding developed through the design and deployment of early Grid applications (e.g. on Globus Toolkit 2) with the emerging Web Services middleware. Their goal was to allow Grid developers to exploit the huge commercial investment in Web Services infrastructure and tooling. The result was the Open Grid Services Architecture (OGSA) [5] – a high-level framework designed to support dynamic virtual organisations that share independently administered data and resources seamlessly across a network of heterogeneous computers. Still under development, this architecture will define the major functional components required to meet those requirements. OGSA-DAI is one of those high-level functional components.

1.3 Databases and the OGSA Grid Environment

It has become increasingly clear that if the OGSA is to support a wide range of communities, then database integration is vital. This paper presents an approach to integrating databases into OGSA so that communities may publicise and share structured data resources, and applications may access and update data stored in them.

There are two main dimensions of complexity to the problem: reconciling implementation differences between server products within a single database paradigm (IBM, Oracle, Microsoft, etc.), and the variety of database paradigms (object, relational, XML etc.). Unsurprisingly, none of the currently deployed database management systems support OGSA Grid integration. Each DBMS is however the result of many hundreds of person-years of effort to provide a wide range of functionality, valuable programming interfaces and tools, and important properties such as security, performance and dependability. As these attributes will be required by applications, we believe that building new Grid-enabled database management systems from scratch is both unrealistic and uneconomic. We have shown that it is also unnecessary, by developing middleware that integrates existing database management systems with the OGSA Grid.

Our intention is that over time, the effort required to achieve this will diminish. As the Grid becomes commercially important, database vendors will embed the middleware functionality directly into their products to provide “out-of-the-box” support for OGSA Grid integration, by directly supporting the emerging OGSA Grid standards, for example, for data access, notification and data dissemination. This simplifies the structure of applications and yields substantial performance gains. Similarly, it is vital that those designing standards for Grid middleware take into account the special requirements to easily integrate databases across a Grid. One of the motivations for OGSA-DAI is to expose and formulate such requirements. Together, these converging developments will reduce the amount of middleware required to integrate databases into the OGSA Grid.

One of the aims, therefore, of the OGSA-DAI project is to explore the requirements and approaches to integrating databases into the OGSA Grid in order to influence both the emerging standards and established database vendors. OGSA-DAI has designed, developed and released a collection of services for integrating database access and description with the core capabilities of OGSA, thus allowing structured data resources to be seamlessly integrated into OGSA Grid applications. We are now gaining feedback from early adopters.

Other work on service-based interfaces to databases is generally more limited in scope or different in context. For example, several relational database vendors support the integration of their products with Web Services (e.g., [6], [7]). Such proposals tend to support the calling of Web Services from within SQL queries, the creation of Web Services from stored procedures, and the publication of Web Services based on database requests. This approach contrasts with, and complements, the development of generic database access services such as those provided by OGSA-DAI. In a Grid setting, the European Data Grid has developed Spitfire [8], a Web Service interface to relational databases for metadata management. OGSA-DAI has adopted a similar approach to Spitfire for authentication, but differs in supporting multiple access languages, asynchronous and third party delivery, as well as an extensible document-based request style interface as opposed to Spitfire's finer grained operations.

There is clearly also a relationship between OGSA-DAI and other data grid functionalities. For example, efficient movement [31] and replication [32] of data are core functionalities in a data grid. OGSA-DAI can make use of existing grid data movement functionalities, as described in Section 4.3, and a data replication service could use OGSA-DAI to read data from or write data to structured data stores. Higher-level grid data management systems, such as SRB [18] or Chimera [33], could use OGSA-DAI either to access structured data resources, or to provide access to their own metadata. As such, OGSA-DAI should be seen as one of a range of components that together support access, sharing, management and coordinated use of data on the Grid.

1.4 Terms used in this paper

The term *database* refers to systems used for managing structured data, such as relational databases and collections of semi-structured files. The term *data resource* is used virtually synonymously with database, the only difference being that it could be considered to include devices, such as telescopes and scanners that produce or accept structured data, or data integration infrastructures that provide the illusion that a single database is being accessed. The term *data source* denotes that subset of data resources that only produce data, e.g. read only data collections. For these two terms, sometimes the word “data” is omitted as it is obvious from the context.

The term *OGSA Grid* refers to a system that complies with the Open Grid Service Architecture [9]. When the word *Grid* is used unqualified it will still refer to an OGSA Grid unless explicitly stated otherwise.

The term *metadata* denotes data that describes either data or a service, normally the former. Sometimes the application or system context determines which data is treated as metadata.

1.5 The Structure of this Paper

The rest of the paper is structured as follows. Section 2 considers the requirements placed upon database integration by OGSA Grid applications. The requirements for, and the architecture of, OGSA-DAI is described in Section 3, in particular detailing one key aspect of OGSA—DAI – the way in which consumers make requests to an OGSA-DAI Grid Database Service. Section 4 describes this aspect in detail. Section 5 describes the mapping of the OGSA-DAI architecture onto the underlying infrastructure, currently OGSi. Finally, the current status and conclusions are presented in Section 6.

The functionalities described in this paper are consistent with those provided by Release 3 of the OGSA-DAI software (July 2003) [10].

2 The Database Requirements of Grid Applications

A typical application that accesses databases may consist of a computation that queries one or more databases and then carries out further analysis on the retrieved data. Therefore, database access should be seen as being only one part of a wider, distributed application. The databases themselves will usually exist independently of any one specific application, in geographically distributed locations. They will be operated under independent and autonomous regimes. If each regime is left to decide independently the access interface with which it presents each database to the OGSA Grid then this has two immediate negative consequences. Firstly, code will need to be written that provides the OGSA Grid interface to their database. This has cost implications for the service provider making the database available to the OGSA Grid. Secondly, those writing OGSA Grid applications or data integration services will have to write code for each proprietary OGSA Grid database interface to which access is required. This has cost implications for those writing the applications. OGSA-DAI provides a substantial part of those requirements thereby avoiding these costs. The argument is, as with any other Grid infrastructure that the higher cost of developing a well-engineered and generic solution is amortised across many applications, the net effect being both an overall reduction in costs and an overall gain in infrastructure quality.

OGSA-DAI addresses both of these issues by implementing a standard, service-based, interface through which databases can be presented to OGSA Grid applications. This standard interface is being specified by the GGF Database Access and Integration Services (DAIS) working group [11]. This approach allows those exposing their databases to the OGSA Grid to reuse code to achieve this, whilst consumers of those services can write code in a way that will be, as far as possible, independent of the specific regime under which the database is administered. In order to ensure that the design of OGSA-DAI achieves this aim, it is first necessary to consider the requirements that it must meet.

To integrate databases into OGSA Grid environments two sets of requirements must be met: firstly, those that deal with the integration of databases into the Grid environment, and secondly those that deal with the functionality offered by databases to applications that must be available via the Grid. These two sets of requirements are considered in turn.

If computational and database components are to be seamlessly combined to create distributed applications then a set of agreed standards will have to be defined and met by all components. While it is too early in the lifetime of the Grid to state categorically what all the areas of standardisation will be, work on existing middleware systems (e.g. CORBA), and emerging work within the Global Grid Forum, suggests that security [12], accounting [13], performance monitoring [14] and scheduling [15] will be important.

Some Grid applications will have extreme performance requirements. In an application that performs CPU-intensive analysis on a huge amount of data accessed by a complex query from

a database system (DBS), then achieving high performance may require utilising high performance servers to support the query execution (e.g. a parallel database server) and the computation (e.g. a powerful compute server such as a parallel machine, or a cluster of workstations). However, this may still not produce high performance, unless the communication between the query and analysis components is optimised. Different communication strategies will be appropriate under different circumstances. If all of the query results are required before analysis can begin then it may be best to transfer all of the results efficiently in a single block from the database server to the compute server. Alternatively, if a significant computation needs to be performed on each element of the result set, then it is likely to be more efficient to stream the results from the DBS to the compute server as they are produced. When streaming, it is important to optimise communication by sending data in blocks, rather than as individual items, and to use flow control to ensure that the consumer is not swamped with or starved of data. Further, it may often be the case that the intended recipient of the result is not the consumer that issued the query. To avoid redundant transmission of data, it should be possible for the query issuer to specify a third -party destination for a result.

We now move beyond considering the requirements that are placed on all Grid middleware by the need to support databases, and consider the requirements that Grid applications will place on the DBMSs themselves. Firstly, there appears to be no reason why Grid applications will not require at least the same functionality, tools and properties as other types of database applications. Consequently, the range of facilities already offered by existing DBMSs will be required. These support both the management of data, and the management of the computational resources used to store and process that data. Specific facilities include:

- query and update facilities
- programming interface
- indexing
- high availability
- recovery
- replication
- versioning
- schema evolution
- uniform access to data and schema
- concurrency control
- transactions
- bulk loading
- manageability
- archiving
- security
- integrity constraints
- change notification (e.g. triggers)

We now consider whether Grid-enabled databases will have requirements beyond those typically found in existing systems. The Grid is intended to support the wide-scale sharing of large quantities of information. The likely characteristics of such systems may be expected to generate the following set of requirements that Grid-enabled databases will have to meet:

2.1 Metadata-driven access

Metadata supports many activities in e-Science that include:

- Management or scheduling through provision of system and administrative information (e.g. defining the performance and capacity of a data resource, the charges and policies governing its use, and current operational state).
- Data discovery or interpretation through provision of data structure and content information, e.g. the schema to which the data conforms and a summary of the content (values) available within that structure. This may be extended with generic information about the interpretation of the schema and content description, e.g. via ontologies [16].

- Resource or access method selection, through indexes or summaries. For example, a database of astronomical objects extracted by processing images may be used as primary data in its own right or as a means of selecting parts of images to reanalyse.
- Data selection or evaluation, to inform human judgements about the data. This varies from pervasive provenance data describing the origin of data to specific annotations suggesting interpretations and implications of the data.

Almost all aspects of metadata can have components that are application specific. Many disciplines are developing extensive bodies of practice concerning the organisation and description of their data. Many applications involve portals, workflows or bespoke code that first examines metadata according to user requirements and then uses this metadata to locate the data, describe which data are accessed, determine what transformations are necessary, to steer analyses and visualisations, and to carry forward information into automatically generated metadata associated with result sets.

It is therefore widely recognised that metadata will be very important for many Grid applications. Currently, the use of metadata in Grid applications tends to be relatively simple – it is mainly for mapping the logical names for datasets into the physical locations where they can be accessed [17] [18]. As users and developers develop more sophisticated applications, and as the Grid expands into new application areas such as engineering and the life sciences, more sophisticated metadata systems and tools will be required. The result is likely to be a *Semantic Grid* [19] that is analogous to the *Semantic Web* [20].

The use of metadata to locate data has important implications for integrating databases into the Grid because it promotes a two-step access to data [21]. In step one, a search of metadata catalogues is used to locate the databases containing the data required by the application. That data is then accessed in the second step. A consequence of two-step access is that the application writer does not know the specific resource that will be accessed in the second step. Therefore the application must be general enough to connect and interface to any of the possible resources returned in step one. This is straightforward if all are built from the same DBMS, and so offer the same interfaces to the application, but more difficult if these interfaces are heterogeneous.

Ideally the two-step approach requires that all resources should provide the same interfaces, but variation in facilities, interfaces and representations is inevitable. Therefore, the variation must be described in metadata that is available via introspection over the resource or service, so that software may be written that adapts to the variations. It also requires that all data is held in a common format, or that the metadata that describes the data is sufficient to allow applications to understand the formats and interpret the data.

A crucial issue is the production of metadata that is reliable. Today most metadata is produced manually. This is both labour intensive and error prone – it will not scale to the envisaged scale of applications and the envisaged number and size of data resources. It is therefore imperative that tools be developed to automatically generate metadata as systems and data collections are built, and to generate data, such as content summaries, by inspecting data resources. OGSA-DAI does not provide such tools but it clearly identifies their necessity and provides a context in which they may be deployed.

Once such metadata is in use additional infrastructure components are required. Primary among these is registries that have matching algorithms relevant to application requirements that operate over this metadata.

2.2 Multiple Database Federation

One of the aims of the Grid is to promote the systematic sharing of scientific data. A recent study of the requirements of some early Grid applications concluded that “The prospect exists for literally billions of data resources, from large curated databases to individual collections,

from major instruments such as the Large Hadron Collider and telescopes to portable instruments, medical scanners, networks of environmental sensors and wearable devices and petabytes of data being accessible in a Grid environment” [22]. Here, data resources include sensors, instruments and wearable devices. If this prospect is realised then it is expected that many of the advances to flow from the Grid will come from applications that can combine information from multiple data sets. This will allow researchers to combine different types of information on a single entity to gain a more complete picture, and to aggregate the same types of information about different entities.

Achieving this will require support for integrating data from multiple data sources, for example through distributed query and transaction facilities. This has been an active research area for several decades, and needs to be addressed on multiple levels. As was the case for metadata-driven access, the design of federation middleware will be made much more straightforward if databases can be accessed through standard interfaces that hide as much of their heterogeneity as possible.

Even when APIs are standardised, there is a higher-level problem of the semantic integration of multiple databases, which has been the subject of much attention over the past decades[23] [24]. In general, the problem complexity increases with the degree of heterogeneity of the set of databases being federated, though the provision of ontologies and metadata can assist. While there is much existing work on federation on which to build, for example in the area of query processing [25], the Grid should give a renewed impetus to research in this area because there will be clear benefits from utilising tools that can combine data over the Grid from multiple, distributed repositories. It is also important that the middleware that supports distributed services across federated databases meets the other Grid requirements. For example, distributed queries run across the Grid may process huge amounts of data, and so the performance requirements on the middleware may, in some cases, exceed the requirements on the individual data resources.

The Grid provides mechanisms with which to build trusted contexts across a number of independent members of a virtual organisation. It also provides mechanisms for moving code as well as data between computational resources within the virtual organisation – either relying on the established trust or using additional safety mechanisms. This means that computation may be moved close to data which often yields savings as code grows much less rapidly than data and the results of many analyses are much smaller than the data examined. Taking advantage of this code mobility poses new planning, scheduling and optimisation challenges which will be aided by a consistent model for computation and data services.

2.3 Summary

In summary, there are sets of requirements that must be met in order to support the construction of Grid applications that access databases. Some are generic across all Grid application components, while others are database specific. Grid applications will require at least the functionality provided by current DBMS. As these are complex pieces of software, with high development costs, building new, Grid-enabled DBMS from scratch is not an option. Instead, new facilities must be added by enhancing existing DBMS, rather than by replacing them. The most commonly used DBMS are commercial products that are not open-source, and so enhancement have to be achieved by wrapping the DBMS externally. Later, we hope that DBMS vendors will choose to integrate grid-enabled data access and integration mechanisms into their code for improved performance, safety and operational convenience.

3 OGSA-DAI Architecture

The goal of OGSA-DAI is to provide a uniform service interface for data access and integration to databases exposed to the Grid, hiding differences such as database driver technology, data formatting techniques and delivery mechanisms. The aim is that through OGSA-DAI, disparate, heterogeneous data sources and resources can be made available as

services that are fully integrated with other OGSA services, for example, for data transport and security. OGSA-DAI services will therefore provide the basic operations that can be used by higher level services to offer greater functionality, such as data federation and distributed queries.

The OGSA-DAI architecture also aims to encouraging the design of efficient applications and evolvable services. To address the former, there is support for grouping multiple requests on an OGSA-DAI service into a single message sent to a service. This reduces latency by increasing the granularity of interactions, and reducing both the number of messages exchanged and the quantity of data transferred. To support service evolution, request messages are self-describing in that they identify the specific operation(s) to be called, so allowing a service to evolve to support new operations without requiring existing applications to be modified.

3.1 Architecture

In the rest of this section we give an overview of the OGSA-DAI architecture, and then explain in more detail how these specific requirements are addressed.

Figure 1 illustrates the principal components within the OGSA-DAI architecture. OGSA-DAI Grid Data Services access databases using drivers, and make use of several additional components, for example, for data formatting, data delivery and request handling. Applications can use the core OGSA-DAI GDS components directly to access individual data stores, or can use a distributed query processor, OGSA-DQP [26], to coordinate access to multiple database services.

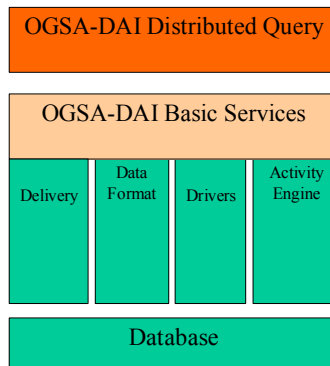


Figure 1: OGSA-DAI components.

The important aspects of the architecture are discussed in the following subsections.

3.1.1 Standardised Interfaces

A benefit that OGSA-DAI brings over and above paradigm-specific database connectivity models like JDBC is a consistent interface regardless of the underlying technology. For example, OGSA-DAI services support access to multiple relational database systems, XML storage managers and file systems. They provide consistent portType definitions and many shared functionalities independently of the database paradigm or specific system being accessed.

As well as query interfaces, OGSA-DAI will evolve to offer interfaces for other common data operations, including transactions and bulk load.

3.1.2 Metadata

OGSA-DAI services provide metadata about the DBMS, e.g. whether it is an Oracle, DB2 or MySQL, etc., DBMS system that is being exposed to the Grid. Also metadata is provided about the capabilities of that DBMS that are being exposed to the Grid through the service interfaces as well as any inherent capabilities of the services themselves, such as third-party delivery and transformations that are available to operate on data retrieved from a database. The connection technology that is employed to connect to the databases is also exposed for clients capable of using such information. For relational databases the database schema may be extracted from the service, which may be helpful to higher level services such as distributed query processing. The metadata may be provided statically, that is when the service is configured, or dynamically which may require additional coding. On the whole the static metadata model is extensible so that communities that employ OGSA-DAI to access databases within a Grid context can provide community specific metadata for the databases they expose to the Grid. The dynamic model can be extended, but this requires more effort from the developers of applications that want to dynamically extend the OGSA-DAI metadata framework.

3.1.3 Sessions

Sessions are important to database systems, for example to relate the queries sent by a consumer to the appropriate transaction context. Therefore, some way is needed to relate a set of requests sent by a consumer (or consumers) over a period of time, where they are part of the same data access session. A DBMS may concurrently support multiple client sessions. For example, RDBMS provide an interface by which clients can connect to the database and, at this point, a new session is created for that client. This allows, for example, multiple concurrent transactions to be run against the same DBMS. Therefore, OGSA-DAI provides a standard way for consumers to create a new session against a data resource. Whilst it would be possible to support this in a platform-independent manner (for example by passing a session id with all requests to a database service), a decision was made to meet this requirement using specific mechanisms provided by OGSI 1.0, as described in Section 5.

3.1.4 Security

It is important to be able to authenticate a consumer so that their request can be executed with the appropriate privileges. As yet there is no consensus within the Grid community as to how authentication should be done within virtual organizations. The aim, however, is to adopt whatever solution emerges from the standardisation efforts currently being undertaken within the Web Services communities. Currently the status quo is to use slightly modified X509 certificates within the Grid community. Hence OGSA-DAI provides a simple authentication layer that consists of a *RoleMap* file and an implementation class that does the role mapping between the Grid credentials used to access the system, which are based on the X509 certificates, and the credentials required to access the underlying DBMS.

3.1.5 Grouping Requests

OGSA-DAI supports a document-oriented interface for database requests, in which a single document may specify a collection of related activities (for example, an update to a database, a query of the updated database, and the delivery of the result of the query to a third party). This increases the granularity of interactions, reducing the number of messages and the quantity of data received. Section 4 describes in detail the structure and capabilities of request documents.

3.1.6 Data Delivery Options

In traditional client, server systems such as JDBC, a client sends a request (e.g. a query) to the database, and receives the result in return. This synchronous request model will not always meet the extreme performance and capacity requirements of some Grid applications. For these, it is important to support a range of data delivery options. For example, the ability to directly deliver the data to a third party, that is to consume the result of the query, or to hold the result at the data service until it is required by another service.

OGSA-DAI supports a range of data-delivery options that can be exploited by Grid applications. Basic requests to an OGSA-DAI Grid Data Service are synchronous: the consumer sends a request document (Section 4) and in return receives a message containing the results. OGSA-DAI also offers asynchronous requests in which the result is not returned to the requestor: it may remain in the Grid Data Service until it is accessed by a consumer, or it may be sent directly to another service. A detailed description of the range of delivery options is presented in Section 4.

4 Perform Documents

The OGSA-DAI *Grid Data Service* (GDS) supports a document-oriented interface for database requests, in which a single document may specify a collection of related activities (for example, an update to a database, a query of the updated database, and the delivery of the result of the query to a third party). This means that the GDS supports few operations - all accesses and updates to the state of a database are carried out by way of a *perform* operation, the parameter of which is a request document. In OGSA-DAI, a request consists of a collection of linked activities, where each activity represents an operation on the service, for example, to query the database, fetch some data from a remote site, etc. The document-based approach has been followed for several reasons. For example, a requirements analysis suggested that a unit of work with a remote database typically involves some database access, data format translation and delivery, and the document approach reduces the number of round trips required in a distributed setting. The request document thus supports composition of activities. It is now described in more detail.

4.1 Request Definition and Execution

The root element of a document, generally referred to as a perform document, sent to the *perform* operation is a `gridDataServicePerform` element. This contains the collection of activities to be performed by the GDS.

4.2 Activities

Each perform document may make use of one or more types of activity. Activities dictate the tasks that a GDS instance can perform on behalf of its client via its enactment engine. An activity is mapped to a Java class that is responsible for acting on or processing the content of that activity. This means that it is straightforward to add new kinds of activities to a GDS, either to provide generally useful functions (e.g., format transformations using an XSLT activity) or operations that are relevant to a specific kind of application (e.g., to compute the codon usage of a DNA sequence). We note, however, that the GDS activity model is not intended to develop into a full workflow language, but is intended to remain of limited expressiveness with a view to supporting common data access and transformation tasks.

The sequence of activity execution within a single request is defined by associating the input of one activity with the output of another activity to define a data flow. For example, Figure 2 illustrates data flows among linked activities using arrows. The *parameter* element gives a name to a literal, which is used as an input to the *sqlQueryStatement*. Where a *parameter* element has no *value*, this must be supplied as an input to the request when it is executed. In this instance it is obtained from a delivery activity, *DeliverToGDT*, which in a GDS is another

portType, the Grid Data Transport portType, allowing communication links to be established between a client and a GDS or a GDS and another GDS. The parameter values, in this case, could be supplied by the client or another GDS. The results are returned to the client synchronously in the response.

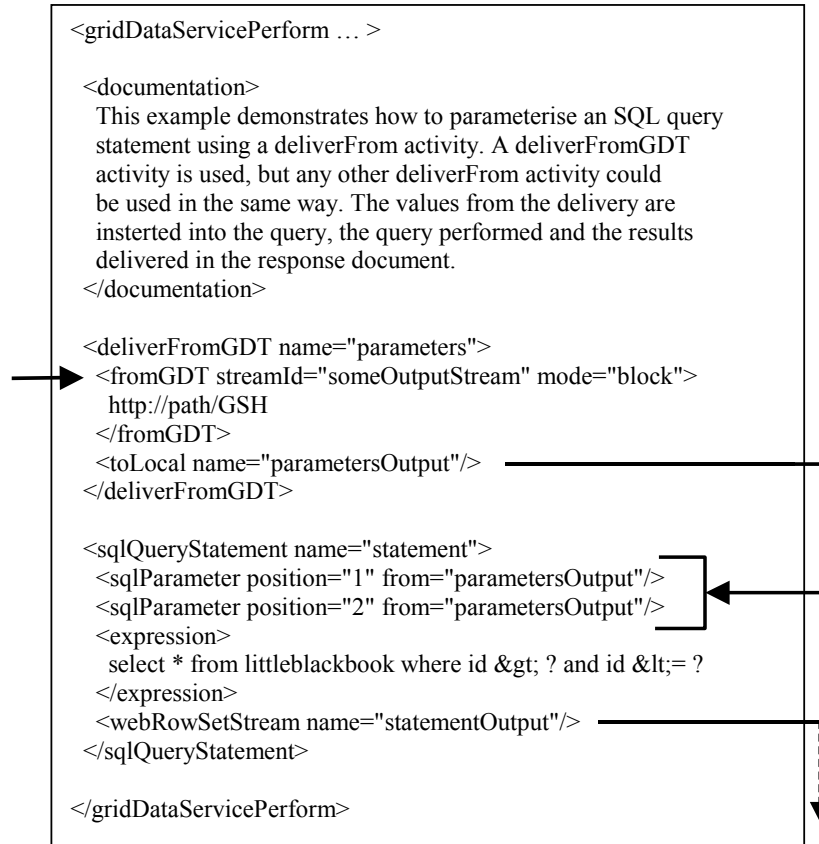


Figure 2 Activity inputs and outputs.

In support of the configurable and extensible nature of requests, concrete activity types supported by a GDS instance are listed in the *ActivityType* service data element (SDE) [27].

Sections 4.1 and 4.2 have described the framework within which concrete activities are defined. The following sections provide some examples of concrete activities.

4.2.1 Relational Database Activities

Activities that provide access to SQL queries, updates and stored procedures are supported. At the time of writing, services have been developed for accessing MySQL, Oracle and DB2 databases. An example *sqlQueryStatement* is shown below. The *sqlParameter* elements define values that are to be passed to the SQL expression. The first element references an activity output value named *chromvalue* from elsewhere in the request, while the second explicitly sets the value. The query is followed by the name by which the outputs are to be made available through the *webRowSetStream* element.

```

<sqlQueryStatement name="statement1">
  <sqlParameter position="1" from="chromvalue"/>
  <sqlParameter position="2">frog</sqlParameter>
  <expression>

```

```

    select * from chromFrag where chrom=? and organism=?
  </expression>
  <webRowSetStream name="statement1result"/>
</sqlQueryStatement>

```

4.2.2 XML Database Activities

Activities are provided for access to XML queries and updates. XPath queries and XUpdate are supported, for implementation over the Xindice database [28]. The elements of *xPathStatementType* are: an optional *collection* element, which identifies the collection over which the XPath query is to be applied; an optional *resourceId* element, which identifies a particular resource in a collection if required; an optional *namespace* element, which defines the namespaces that are used in the XPath expression; *expression* which contains the XPath expression; and *resourceSetStream* which is used to identify the result sequence.

The following shows a sample document for a simple query to an XML data service, retrieving information on chromosome fragments of length less than 20000.

```

<xPathStatement name="chromStatement">
  <expression>/chromFrag [@len<20000]</expression>
  <resourceSetStream name="chromStatementOutput"/>
</xPathStatement>

```

The result of the query is made available to other activities through the *resourceSetStream* label, which is defined as an activity output. This output can then be used, for example, as input to a delivery, as described in the next section.

4.3 Delivery Activities

The ability to deliver data to and/or from a GDS is an intrinsic part of this service. From the client's perspective, the operation of a GDS is either synchronous or asynchronous.

In the synchronous case, the client submits a perform document to *GDS::perform* and expects the results of the complete request in response to this call. Thus data delivery is considered to be part of the response to the *perform* operation call. The default delivery is to in-line the results in the response to the client unless instructed otherwise by an explicit delivery activity in the perform document. This is illustrated in Figure 3: a consumer sends a request containing a query to a GDS, and receives the results back directly.

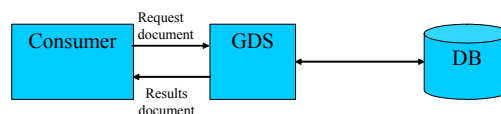


Figure 3 Synchronous delivery of query results.

The following *gridDataServicePerform* document to a relational GDS defines a request that queries a single table using SQL, and returns the result to the analyst in the response.

```

<gridDataServicePerform ... >
  <sqlQueryStatement name="statement">
    <expression>
      select * from chromFrag where len < 20000
    </expression>
    <webRowSetStream name="statementresponse"/>
  </sqlQueryStatement>
</gridDataServicePerform>

```

In the asynchronous case, the client submits a perform document to *GDS::perform* and expects the GDS to complete the processing of the request after the *perform* operation has

returned control to the client. This is particularly important where long running requests are submitted. Data delivery in this case follows either an active or a passive approach.

In the active case, the perform document must contain enough information for the GDS to interact with delivery protocols or services in order to deliver the data. In the passive case, the GDS waits for external services or protocols to initiate the data delivery. The passive case is represented by the *inputStream* and *outputStream* activities. The following delivery activities are defined within OGSA-DAI:

Activity	Usage
<i>deliverFromGDT</i>	Define input configuration depending on the protocol chosen (GDT/GFTP/URL). Also define an output element that an internal activity can reference in order to obtain the data that this activity will retrieve. GDT stands for Grid Data Transport, and GFTP stands for Grid FTP, as explained below.
<i>deliverFromGFTP</i>	
<i>deliverFromURL</i>	
<i>deliverToGDT</i>	Define an input element that references an output element in some other activity defined in the perform document to which the data will be delivered. The defined output configuration depends on the protocol chosen (GDT/GFTP/URL).
<i>deliverToGFTP</i>	
<i>deliverToURL</i>	
<i>inputStream</i>	Defines an output element that an internal activity can reference in order to obtain data when data is presented to this activity by an external service.
<i>outputStream</i>	Defines an input element that references an output element in one other activity defined in the <i>request</i> document. This provides the data that this activity will send in response to any request by an external service.

The underlying transport protocol used to support such an activity is provided by the bind element of the WSDL.

In the above, *deliverToGDT* and *deliverFromGDT* involve the use of the *Grid Data Transport* (GDT) portType. This portType acts as an endpoint for data delivery, and must be supported by the Grid Service that is the target of this delivery activity. Thus, for example, a GDS service *S* can deliver to a remote service *R* using *deliverToGDT* only if *R* supports the GDT portType. *R* may or may not be a *GDS*. The *GridDataTransport* portType defines four operations *getFully*, *getBlock*, *putFully* and *putBlock*.

GridFTP [29] is the principal data transport service used on the Grid. The activity type *deliverFromGFTP* allows data to be received by a GDS from an external location using GridFTP, and *deliverToGFTP* allows data to be delivered by a GDS to an external location using GridFTP.

The most straightforward asynchronous scenario for a GDS is that a requester executes a request containing a query to a GDS, the result of which is delivered to a file, in this case using GridFTP. This scenario is illustrated in Figure 4.

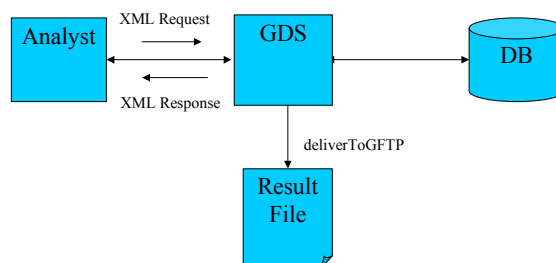


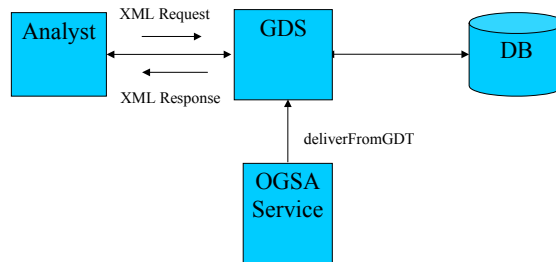
Figure 4 Asynchronous delivery of query results to a file.

The following perform document defines a request containing the same query as in Figure 3, but uses GridFTP to deliver the result to a (potentially remote) file. The response returned to the analyst indicates that the request has been received, and is queued for evaluation. We note that previous database access middleware, such as JDBC, has tended not to provide this sort of indirect delivery capability.

```
<gridDataServicePerform ... >
  <sqlQueryStatement name="statement">
    <expression>
      select * from chromFrag where len &lt; 20000
    </expression>
    <webRowSetStream name="statementresponse"/>
  </sqlQueryStatement>

  <deliverToGFTP name="d1">
    <fromLocal from="statementresponse"/>
    <toGFTP host="ogsdai.org.uk"
      port="8080" file="path/to/myfile.txt"/>
  </deliverToGFTP>
</gridDataServicePerform>
```

There are many situations in which a GDS might usefully interact directly with another service. For example, it might be useful for the results of a query over one GDS to be conveyed to another GDS by a replication infrastructure. By contrast, Figure 5 illustrates a scenario in which a GDS is to be used to update a database using data delivered to it from another service. In this example, it is assumed that the other service implements the GDT portType introduced in Section 4.3. The *deliverFromGDT* activity will invoke the *getFully* operation on the related service to obtain the required data for update. This data is then used as the parameter of an *sqlUpdateStatement* activity that inserts the retrieved value into the *chromFrag* table.

**Figure 5 Asynchronous delivery of results from a service.**

```
<gridDataServicePerform ... >
  <deliverFromGDT name="d1">
    <fromGDT streamId="otherrequestasynch/d1"
      mode="full">
      http://ogsdai.org.uk/GDTService/my/GDT/GSH
    </fromGDT>
    <toLocal name="datatoininsert"/>
  </deliverFromGDT>

  <sqlUpdateStatement name="statement">
    <sqlParameter position="1" from="datatoininsert"/>
    <expression>insert into chromFrag values ?</expression>
  </sqlUpdateStatement>
</gridDataServicePerform>
```

5 Mapping OGSA-DAI onto OGSI v1.0

Whilst the concepts embodied by the OGSA-DAI design are largely technology independent (they assume only that applications are built from services that communicate by exchanging messages) the version available at the time of writing is built on top of the Open Grid Services Infrastructure (OGSI) v1.0. To allow the definition and implementation of OGSA it was necessary to define a common infrastructure that would provide those functional components with a uniform operating context mapped onto the underlying web services specifications. The Open Grid Services Infrastructure (OGSI) was the first specification of such an infrastructure [27]. This section introduces OGSI, and then describes the mapping of OGSA-DAI onto it.

5.1 Introduction to OGSI v1.0

The Grid Service Specification [27] defines how Grid Services can be described using WSDL portTypes. Clients and other Grid Services may interact with a Grid Service via the operations provided through its portTypes. Different types of Grid Service can be realised by aggregating different portTypes. Service data elements are used to provide access to the state, properties and capabilities of a Grid Service. Each portType has an associated set of service data elements. A service can also have service data that is independent of its portTypes.

The Grid Service Specification defines a core set of portTypes that describe a set of behaviours expected from different types of Grid Service. These portTypes include:

- *GridService* – a standard portType that is required to be available in all Grid Services. Functionality supported includes service description, through service data elements, and lifetime management.
- *Factory* – for Grid Services that create other Grid Services.
- *ServiceGroup*, *ServiceGroupEntry*, *ServiceGroupRegistration* – a set of interfaces that allows Grid Services to register with other Grid Services.
- *NotificationSource* – an interface for Grid Services to accept subscriptions from other services that would like to receive notifications of events of interest, such as state changes in the service.
- *NotificationSink* – for Grid Services that can receive notifications from services implementing the NotificationSource portType.

Each instance of a Grid Service is identified by means of a:

- *Grid Service Handle* (GSH) – a handle that identifies where and how the service can be accessed, and which is constant for the lifetime of the Grid Service. A GSH must be a valid URI.
- *Grid Service Reference* (GSR) – a detailed description of the Grid Service including the portTypes it implements, the operations it supports and its physical address on the network. Unlike a GSH, the GSR can change over the lifetime of a Grid Service.

5.2 The OGSA-DAI to OGSI v1 mapping

This release of OGSA-DAI has been developed over the Globus Toolkit 3 implementation of OGSI v1.0, which is available from www.globus.org. The *Grid Data Service* (GDS) is the primary OGSA-DAI service. A *Grid Data Service* supports data access through the *GridDataService* portType and data delivery through the *GridDataTransport* (GDT) portType. A GDS is a transient service that supports the interaction with a single client with a single database. In essence, a GDS might be considered to represent a session with a database.

Thus a GDS manages an authenticated collection of requests to a resource, and stores state of use to the requester, for instance, on the status and results of requests.

An instance of a GDS is created using a *Grid Data Service Factory* (GDSF). The GDSF itself may be located using a *DAI Service Group Registry* (DAISGR)

Figure 6 depicts the basic services and portTypes described by OGSA-DAI. The components shown on this diagram describe the typical state of a running OGSA-DAI system. Some of the components are taking part in a simple OGSA-DAI scenario where an analyst wants to perform an SQL query across a dataset with a know name and schema. This scenario is not intended to capture the full scope of function that OGSA-DAI is intended to provide but to introduce the components in context.

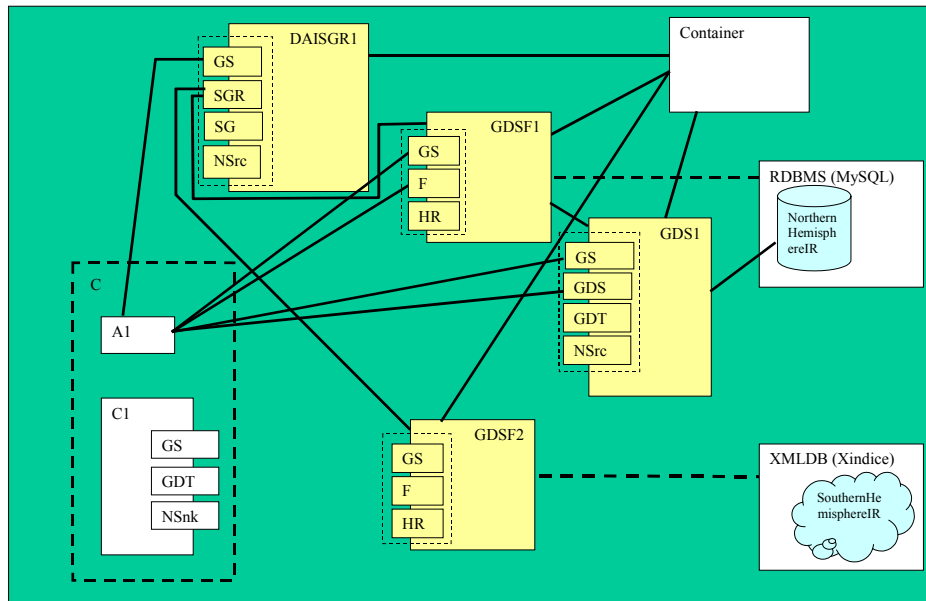


Figure 6: Grid Data Service Operation.

The steps are as follows:

1. The Container, in which all Grid services run, starts. During this process, the Container creates the persistent DAI Service Group Registry DAISGR1. The Container also creates the persistent Grid Data Service Factories GDSF1 and GDSF2. Each Factory instance is responsible for registering itself with the DAISGR1.
2. The analyst A1 Starts and locates the registry instance DAISGR1 that has been created in the container. The Container is not responsible for A1. A1 and the Container are likely to be running on different physical machines in separate organizations.
3. A1 uses the registry DAISGR1 in order to discover a factory GDSF1 that supports the required database and statement type for retrieving the data using an SQL query.
4. A1 uses GDSF1 to create a grid data service GDS1 with the appropriate configuration, as described in more detail below.
5. A1 formulates a script containing a query and passes this to GDS1. GDS1 contacts the database holding the required data and, in this case, returns the data in the response. The OGSA-DAI scope also covers the case where result data is delivered to a third party consumer C1 using the mechanisms supported by the *Grid Data Transport* portType (*GDT*), as described in Section 4.
6. A1 removes GDS1 (or the service instance times out).

The portType acronyms are as defined in the previous section. Other terminology used in the diagram is expanded in the tables below.

Service Instances	
DAISGR1	DAIServiceGroupRegistry service instance.
GDSF1, GDSF2	GridDataServiceFactory service instances configured to build <i>Grid Data Services</i> that act as proxies for either a specific MySQL RDBMS or a specific Xindice XML DBMS (this factory is not used in the simple scenario described but it still exists as a service instance for use when required).
GDS1	GridDataService service instance created and configured to interact with a specific database within the MySQL database management system.
C1	Consumer service instance. This system design does not discuss the service type of C1 other than that it must implement the <i>GDT</i> portType and possibly the <i>NSnk</i> portType. The service type of C1 is a client side consideration and is outside the scope of OGSA-DAI.

Table 1: Service Instance

Other Terminology	
A1	Analyst. The application, or other client, that is making requests to GDS1 . The term Analyst is used as in many e-Science experiments an “Analyst” will prepare requests for data to be retrieved and have the data delivered to some third party process, or Consumer , for processing.
C	Consumer is generally the destination of the data

	<p>produced by GDS1. The consumer may not be a grid service like C1 that supports an interface defined by portTypes. In this case, the OGSA-DAI mechanisms for data transport must be able to support data transport to targets that are not OGSI compliant services.</p> <p>The Consumer is shown as a dotted line as it could be the analysts process A, some third party or some other Grid service (C1).</p>
RDBMS	Relational Database Management System.
NorthernHemisphereI R	Some application data held in a relational database.
XMLDB	XML Database Management System.
SouthernHemisphereI R	Some application data held in an XML database.

Table 2: Other Terminology

A1 and **C1** are commonly referred to as the client in this document.

Using the *perform* operation of the GridDataService portType of the GDS, the analyst submits a request that a query be run and the results delivered. The example illustrates the results of the request being returned to the user, but other scenarios are possible, as was illustrated in Section 4.

In the example, the service GDS1 is created by a *Grid Data Service Factory* (GDSF), which is a specialised factory service that can create new *Grid Data Services* (GDS). A GDSF can create GDSs of varying types, and a client can query the GDSF to determine the available configurations for the GDSs. A client can then request that the GDSF create a GDS with the configuration of most relevance to the client. For example, the following XML fragment illustrates the configuration data of a GDSF that can (potentially among other things) create GDSs that provide access to a *frogGenome* database.

```
<dataResource ...>
  <!-- Information about Activity mapping goes here -->
  ... activity mapping goes here ...

  <!-- Metadata describing the data resource -->
  <dataResourceMetaData>
    <!-- These elements and their contents are optional -->
    <productInfo>
      <productName>MySQL</productName>
      <productVersion>4</productVersion>
      <vendorName>MySQL</vendorName>
    </productInfo>
    <relationalMetaData>
      <databaseSchema callback="... class to extract DB schema ..."/>
    </relationalMetaData>
  </dataResourceMetaData>

  <!-- Rolemapper to use for mapping from
  Grid credentials to database roles -->
  <roleMap name="Name"
    implementation="... some class ..."
    configuration="examples/RoleMap/ExampleDatabaseRoles.xml"/>

  <!-- Data resource details.
  These settings allow GDS to connect to the DBMS -->
  <driverManager driverManagerImplementation=
```

```
        "... driver manager class ...">
    <driver>
      <driverImplementation>
        org.gjt.mm.mysql.Driver
      </driverImplementation>
      <driverURI>
        jdbc:mysql://localhost:3306/frogGenome
      </driverURI>
    </driver>
  </driverManager>
</dataSourceConfig>
```

A GDSF allows clients to access information about its state, including information on the GDSs it can create, such as locations, drivers and platforms through its *GridService::findServiceData* operation.

5.3 Future Mappings onto Grid Services

Experience with several early implementations of OGSI, and a desire to achieve greater synergy between Grid and Web Service developer communities, has resulted in a recent proposal to revise the mapping onto Web Services. This is in the form of a set of specifications called WS-Resource Framework (WSRF) [30] which deliver a similar infrastructure, but with an explicit separation between stateful resources and the web services which present mechanisms for using those resources; the WSRF provides a means of accessing the state of resources. As OGSA-DAI has a relatively simple operational model, it should be relatively straightforward to map it onto whatever emerges as the preferred way of building Grid applications.

6 Conclusions

With a view to exploring how best to integrate databases and the Grid, the OGSA-DAI project has developed a collection of Grid Database Services, key features of which are as follows:

- They provide consistent access to multiple database paradigms; this paper has described activities that support access to both Relational and XML database systems.
- They support flexible data delivery, providing facilities for incremental and bulk delivery, to and from services or files, in a synchronous or asynchronous manner.
- They are fully integrated with existing Grid functionalities for authentication and data transport.
- They are tracking the ongoing standardisation activity within the *Database Access and Integration Services* (DAIS) Working Group of the Global Grid forum (www.gridforum.org).
- They are tracking emerging Grid Service standards, with a version having been released that is compatible with OGSI version 1.0, making use of OGSI portTypes for service creation, service description and lifetime management.

Many of the advances to flow from the Grid will come from applications that can combine information from multiple data sets. This will allow researchers to combine different types of information on a single entity to gain a more complete picture, and to aggregate the same types of information about different entities. Achieving this requires support for integrating data from multiple DBS, for example through distributed query facilities. The design of federation middleware is made much more straightforward by OGSA-DAI providing standard interfaces that hide as much of the heterogeneity as possible. The OGSA-DQP [26] system builds a distributed query processing service on top of a set of OGSA-DAI database services. The query operations (filter, join, etc.) are distributed over nodes acquired from the Grid, so

as to distribute the work of query evaluation in a scalable manner – the system can also exploit parallelism in the database operators and other operations invoked from a query, to provide scaleable performance. OGSA-DQP exploits the standard OGSA-DAI interfaces to extract metadata about data held behind a service (e.g. its size, the availability of indices, etc.) for optimisation. Having determined the best way to evaluate a query across each of the database services, it exploits the standard interfaces to send query requests to all the databases, while processing their responses is simplified by OGSA-DAI's common query result format. Without OGSA-DAI, the DQP service would have to be designed to interface directly with a set of heterogeneous DBMS – a much harder task.

Overall, we believe that the OGSA-DAI system described in this paper is the most comprehensive attempt to date to provide generic database access capabilities within a Grid setting. The design seeks to provide enhanced functionality compared with other database middleware by supporting access to multiple database paradigms and by providing flexible data delivery both to and from the database services. The OGSA-DAI software (along with OGSA-DQP) is freely available in open-source form from www.ogsadai.org.uk.

Acknowledgements: This work is supported by the UK e-Science Core Programme, the EPSRC and the DTI, whose support we are pleased to acknowledge.

Trademarks:

IBM and DB2 are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both.

Oracle is a trademark of Oracle Corporation

MySQL is a trademark of MySQL AB in the United States and other countries.

Globus Project and Globus Toolkit are trademarks of the University of Chicago.

References

- [1] The Globus Toolkit V2.4, <http://www.globus.org/gt2.4/download.html>
- [2] P. Dinda and B. Plale, A unified relational approach to Grid information services, Technical Report, Global Grid Forum, 2001, GWD-GIS-012-1
- [3] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, Web Services Architecture, W3C Working Group Note, 2004, <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211>
- [4] S. Chatterjee, J. Webber, Developing Enterprise Web Services: An Architects Guide To Best Practices, Penguin Books, 2003, ISBN 0131401602
- [5] I. Foster, C. Kesselman, J. Nick, Steven Tuecke: Grid Services for Distributed System Integration. IEEE Computer, 35(6): 37-46, (2002).
- [6] S. Malaika, C.J. Nelin, R. Qu, B. Reinwald and D.C. Wolfson, DB2 and Web Services, IBM Systems Journal, 41(4), 666-685, 2002.
- [7] T. Van Raalte and R. Ward, Oracle 9i Application Server Web Services Developer's Guide, Oracle Corporation, 2002.
- [8] W. H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. McCance and M. Silander, Project Spitfire – Towards Grid Web Service Databases, Presented at Database Access and Integration Services Working Group, 5th Global Grid Forum (GGF5), Edinburgh, 2002.

- [9] I. Foster, D. Gannon, H. Kishimoto – Editors, The Open Grid Services Architecture, GGF OGSA Working Group, 2003, <http://forge.gridforum.org/projects/ogsa-wg/document/draft-ggf-ogsa-spec/en/13>
- [10] OGSA-DAI V3, <http://www.ogsadai.org/releases/ogsadai.php#R3>
- [11] Global Grid Forum – Database Access and Integration Services Working Group, <http://forge.gridforum.org/projects/dais-wg/>
- [12] Global Grid Forum – OGSA Security Working Group, <http://forge.gridforum.org/projects/ogsa-sec-wg/>
- [13] Global Grid Forum – Accounting Models Research Group, <http://forge.gridforum.org/projects/acct-rg/>
- [14] B. Tierney, R. Aydt, et. Al. A Grid Monitoring Architecture, Global Grid Forum 2002
- [15] Global Grid Forum – Scheduling and Resource Management Area, <http://forge.gridforum.org/projects/srm/>
- [16] C. Wroe, R. Stevens, Carole A. Goble, A. Roberts, M. Greenwood, A Suite of Daml+Oil Ontologies to Describe Bioinformatics Web Services and Data. *Int. J. Cooperative Information Systems*, 12(2): 197-224, 2003.
- [17] M. Ripeanu, I. Forster. A Decentralized, Adaptive, Replica Location Service. 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)Edinburgh, Scotland, July 24-16, 2002.
- [18] A. Rajasekar, M. Wan, R. Moore: MySRB & SRB: Components of a Data Grid, 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), 301-310, 2002.
- [19] D. De Roure, N. Jennings, et. Al (2001). Research Agenda for the Semantic Grid : A Future e-Science Infrastructre, UK National e-Science Centre : UKeS-2002-02.
- [20] T. Berners-Lee, J. Hendler, et. al. (2001), The Semantic Web, *Scientific American*, May 2001.
- [21] P. Watson, “Databases and the Grid”, in *Grid Computing*, ed. Berman, Fox, Hey, Pub. Wiley, April 2003, ISBN 0470853190.
- [22] D. Pearson, Data Requirements for The Grid: Scoping Study Report, UK Grid Database Taskforce.
- [23] A.P. Seth, J.A. Larson (1990), Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, *ACM Computing Surveys* 22(3): 183-236
- [24] T. Lahiri, S. Abiteboul, et. Al. (1999), Ozone : Integrating Structured and Semistructured Data, Seventh Int. Workshop on Database Programming Languages, Kinloch Rannoch, Scotland
- [25] C.T. Yu, C.C. Chang (1984), Distributed Query Processing, *ACM Computing Surveys* 16(4) : 399-433
- [26] N. Alpdemir, A. Mukherjee, N.W. Paton, P. Watson, A.A.A. Fernandes, A. Gounaris and J. Smith, Service-Based Distributed Query Processing on the Grid, 1st Intl. Conference on Service-Oriented Computing (ICSOC), 467-482, Springer-Verlag, 2003.
- [27] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt and D. Snelling , Open Grid Services Infrastructure (OGSI) Version 1.0, Global Grid Forum Recommendation, Jun. 2003.
- [28] Apache Xindice, <http://xml.apache.org/xindice/>
- [29] Globus GridFTP, <http://www.globus.org/datagrid/gridftp.html>
- [30] D. Ferguson, I. Foster, J. Frey, S. Graham, F. Leyman, M. Nally, T. Storey, S. Tuecke, S. Weerawarana, (2004), Modeling Stateful Resources with Web Services
- [31] R.K. Madduri, C.S. Hood, W.E. Allcock: Reliable File Transfer in Grid Environments, 7th Annual IEEE Conference on Local Computer Networks (LCN), 737-738, 2002.
- [32] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I.T. Foster, B. Tierney: File and Object Replication in Data Grids, 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), 76-86, 2001.

- [33] I.T. Foster, J.-S. Vöckler, M. Wilde, Y. Zhao: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. Proc.14th International Conference on Scientific and Statistical Database Management (SSDBM), 37-46, 2002.