Database Access and Integration Services on the Grid

Norman W Paton
Department of Computer Science,
University of Manchester,
Manchester, M13 9PL, UK.
email: norm@cs.man.ac.uk

Malcolm P Atkinson
National e-Science Centre,
15 South College Street,
Edinburgh EH8 9AA, UK.
email: mpa@nesc.ac.uk

Vijay Dialani
Dept. of Electronics and Computer Science,
University of Southampton
Southampton SO17 1BJ, UK.
email: vkd00r@ecs.soton.ac.uk

Dave Pearson
Oracle UK Ltd., Thames Valley Park,
Reading RG6 1RA, UK
email: dave.pearson@oracle.com

Tony Storey
IBM United Kingdom Laboratories,
Hursley Park, Winchester, SO21 2JN , UK
email: tony_storey@uk.ibm.com

Paul Watson
Department of Computing Science,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK
email: Paul.Watson@ncl.ac.uk

February 1, 2002

Abstract

Research and development activities relating to the Grid have generally focused on applications where data is stored in files. Although this has allowed progress to be made rapidly with various aspects of Grid infrastructure, database management systems have a central role in data storage, access, organisation, authorisation, reorganisation, etc, for numerous applications, including those in e-Science. This document makes a proposal for a collection of services that support: (i) consistent access to databases from Grid applications; and (ii) coordinated use of multiple databases from Grid middleware. It is hoped that the proposal, which should be considered to be preliminary, can help to foster a wider activity on the formulation of standards for databases and the Grid.

1 Introduction

Many scientific applications use database management systems to structure and store important data sets. Many such applications stand to benefit from Grid facilities for supporting the sharing, dissemination and analysis of such data sets. However, research and development activities relating to the Grid have generally focused on applications where data is stored in files. Our analysis of the requirements of e-Science projects [Pea02] has shown that there is an urgent and widespread need for the interconnection of pre-existing and independently operated databases. This document seeks to encourage the development of standardised facilities that can meet this need.

In particular, a proposal is made for the staged development of a collection of Grid database services. In so doing, the intention is to encourage wider discussion and interaction on the most appropriate services for use with

databases in a Grid setting.

The proposed services provide generic database-oriented functionalities, such as, allowing queries and updates to be evaluated over a database, and transactional coordination of accesses to one or more databases. As service definitions essentially state *what* functionality is to be supported, and not *how* that functionality is provided, a single service may come to be implemented in different ways. This, for example, allows alternative implementations with different performance characteristics, but also allows the provision of *virtual* services. For example, a *vitrual* database service might provide the illusion that a single database is being accessed, whereas in fact, the service is accessing several databases using distributed query processing techniques. The overall focus, however, is on services that allow access to and integrated use of existing, autonomously managed databases within Grid middleware.

The adoption of a service-oriented approach is intended to allow the development of composable components, where individual categories of service provide consistent and complementary behaviour, while hiding implementation details. Databases already exist and are being developed and used within Grid settings without reference to standards. The hope is that this document can move forward the process of identifying core services, implementations of which will allow effective integration of database systems into Grid middleware. We note that service-based approaches seem to be on the ascendency, with both Web Services [Kre01] and the Open Grid Service Architecture (OGSA) [FKNT02] promoting service-based approaches. This document casts its service descriptions within the framework provided by the OGSA, which in turn exploits the Web Services Description Language [CCMW01].

The proposal presented in this document is preliminary; its purpose is to initiate discussion and identify groups and individuals that would be interested in contributing to the development of successors.

How does this document relate to other work on Databases and the Grid? There are existing proposals for storage resource brokers (e.g. SRB [BMRW98]) and database connectivity facilities (e.g. Spitfire [H$^+$01]) within a Grid setting. In this document we initiate a search for a generic framework for composable services that will support a wide range of database use by grid services and grid applications, and which will facilitate the integration of data from multiple sources. These sources can be either transient or persistent database services, where persistent services include those operated independently of the grid infrastructure.

The document is structured as follows. Section 2 makes some remarks on the scope and emphasis of the proposal. Section 3 introduces relevant terminology and states some assumptions relating to services and to databases. Section 4 introduces the proposed database services, and discusses some issues relating to their implementation. Section 5 describes how the proposed services might be included within the Open Grid Services Architecture (OGSA) [FKNT02], and Section 6 indicates how the proposed services can be used to provide access to different kinds of resource. Section 7 identifies some issues relating to the integration of databases services with the OGSA. Section 8 describes how the proposed services could be developed in an incremental manner. Section 9 presents some conclusions.

2 Scope of Proposal

This document makes a proposal for the staged development of a collection of Grid database services. As such, its purpose is to encourage the adoption of standards for integrating database systems into Grid middleware. Such an activity is only likely to be successful if there is widespread input into the development of the proposal, and all aspects of the contents of this document should be considered to be preliminary. An intention in making a reasonably concrete proposal at this stage is in part to elicit responses, which can be used to guide the development of better proposals in the future.

This document should be seen as complementary to [Wat01]. The latter provides an overview of issues associated with the integration of Grid and database technologies, and suggests a service-based approach. This document develops that theme.

The proposal has several characteristics:

1. It is largely independent of any specific Grid toolkit. There are several reasons for this:
    - Grid toolkits are evolving; presenting a database proposal in the context of a specific

version of a particular toolkit could skew the proposal, complicate evolution of the document and restrict the community that was inclined to contribute to the proposal.

- Incorporating databases into a Grid middleware is likely to impact on other middleware services. For example, database metadata is sure to impact on the resource discovery functionalities of an existing middleware. The full consequences of such changes would be difficult to address within a database-centred document.
- Service descriptions by their very nature tend to be amenable to implementation using different protocols and runtime environments.

2. It does not propose the development of a new database management system (DBMS) for the Grid. The principal focus on connecting to existing systems hopefully does not require explicit justification. Thus, supporting the services described in this document, for the most part, involves wrapping[1] existing systems so that they adhere to a consistent interface. Such wrapping need not, in general, require changes to be made to the wrapped DBMS, although direct implementation of the services by a DBMS developer would provide performance gains.

   Certain of the functionalities described later, such as distributed transaction management or distributed query processing, span multiple databases. Such functionalities cannot be obtained purely by wrapping individual databases, but would require the development or deployment of distributed transaction managers or query processors as services within a Grid setting.

3. It is independent of any specific data model or database access language. This is perhaps a bit more controversial. We make a few motivating observations at this stage:

   - Many aspects of database connectivity are data model independent. For example, authorisation and transaction models are independent of the data model being accessed.
   - Scientific data is stored in many different formats, not all relational, and both XML and object models are widely used.

Few of the services discussed in this document are particularly novel. The purpose of the document is to encourage the consistent integration of existing database and database-oriented functionalities into a Grid environment.

3 Context

This section sets the scene for the remainder of the paper by defining relevant terms and assumptions relating to Web Services and to databases.

3.1 Services

A *service* is a network enabled entity that provides some capability through the exchange of messages [FKNT02]. Web Services standards [Kre01] specify facilities for describing, discovering and using services. One of the Web Services standards is the Web Services Description Language (WSDL) [CCMW01], which provides an XML model for describing Web Services. Of particular relevance to this document, a WSDL description of a service includes specifications of:

Types: the data type definitions that describe the data to be exchanged.

Messages: the definitions of the data to be sent to or from the service.

PortTypes: the operations that are supported by the service. For each operation this includes its *name*, *input* and *output* messages, and *fault* messages.

In essence, these describe *what* capabilities a service provides. A WSDL definition also provides information on *where* the service is and *how* (i.e. using what protocol) messages can be sent.

A WSDL service definition is an XML document. A service definition can *import* other WSDL documents, and thus a service definition can be composed using existing types, messages and port-types. In this document,

[1] In this document the term *wrapper* is not intended to imply that the only way to access a resource is through a wrapper, but rather that the wrapper provides a well defined interface through which a resource can be accessed.

we use the term *interface* to refer to a collection of types, messages and port-types that describe a specific capability. Thus a service can support the capabilities described in one or more interfaces.

The *Open Grid Services Architecture* (OGSA) is a collection of web services definitions. These provide (i) abstract descriptions of existing Grid capabilities using WSDL; and (ii) collections of standard interfaces that characterise Grid services, for example, addressing the lifetime of services, authentication, etc. As such, the OGSA can be held to provide an organisational framework into which additional services can be slotted. Thus, by choosing to conform to and implement the standard interfaces, software components can be integrated into the OGSA. This document makes a proposal for a collection of database services within the context of the OGSA.

3.2 Databases

A *database service* can be defined as any service that supports a database interface. As such, there is no requirement imposed in this document that a database service provides persistence or is directly implemented using a database management system. Service interfaces are abstract and are not prescriptive in terms of how their capabilities should be supported. Furthermore, in this document we are also not prescriptive in terms of the data model that underpins a database service. Thus specific database services could provide access to relational databases, object databases, XML repositories or specialist storage systems (e.g. for storing matrices or genome sequences).

The principal service considered in this document is referred to as a *Grid Database Service* (GDS). This service provides capabilities for querying, updating and evolving a database. However interfaces are also described for:

Data Delivery: providing facilities for transmitting structured data. Such a service differs from file transfer mechanisms in that the internal structure of the data can be taken account of within the service.

Transactions: providing facilities for coordinating collections of operations, for example, with a view to controlling interference between concurrent users.

Database Metadata: providing access to information about the capabilities of a database service and the data it provides access to.

We note that both *Data Delivery* and *Transaction* capabilities are of relevance whether or not databases are being used. Thus a data delivery service could be used to deliver structured results from a real-time data feed to a laboratory information management system, and transactions can be used to coordinate accesses and updates to any software system implementing appropriate interfaces.

The provision of a grid database service within the context of the OGSA involves (i) describing interfaces that extend those already provided as part of the OGSA; (ii) delimiting the services that import those interfaces; and (iii) describing how those services fit within the framework provided by the OGSA. Section 4 covers point (i), and Section 5 addresses points (ii) and (iii).

4 Database Services

This section introduces the functionality of the proposed database services. The description provided in this section is largely independent of the OGSA, and could thus be associated with database service provision in web services in general. Indeed, as Web Services develop, it is likely that proposals will emerge for standard interfaces to databases, which might usefully be tracked by standardisation activities relating to Grid services.

Interfaces are described in this section at a somewhat more abstract level than in WSDL. Future versions of the document will include WSDL definitions for all the interfaces described.

4.1 Database Discovery

In a services architecture, a service provider publishes a description of a service to a service registry. This registry can then be consulted, by a service requestor, an appropriate service description extracted, and finally a binding created that allows calls to be made to the service by the requestor [Kre01].

This means that descriptions of database services, like other services, must be able to be published. A basic service description that might be published would be the

WSDL of the service. In the case of a database service, it might be useful to publish substantial information on the contents of the database, in addition to details of the operations that the database service supports. The discussion of precisely what details might be published about a database and how is beyond the scope of this document. It is assumed in what follows that a registry lookup has taken place that returns a *Grid Service Handle* (GSH), a globally unique name for each service instance [FKNT02].

4.2 Database Statements

Database statements allow queries, updates, loads or schema change operations to be sent to a database system for execution. This implies that the database system over which a service is being provided supports a query or command language interface. This is certainly true for relational databases, but is less uniformly the case for object databases or XML repositories. As such, this is a point of tension with the principle that the proposal should be data model independent.

The basic operations that form the *database service interface* are given in Table 1. *IN* denotes an input parameter, *OUT* denotes an output parameter, and *OPTIN* and *OPTOUT* denote optional parameters. The operations on a GDS will be atomic – they will either execute completely or the GDS will behave as if they had never been invoked. As GDS operations often involve the transfer of large amounts of data and the use of secondary storage, they may take a substantial amount of time to execute. As a consequence, they are prone to interruptions due to various failures. The implementations of all database services must handle such failures in order to achieve atomicity. Were this not the case, the state of the database accessed through a service could become indeterminate, making it impossible for clients to continue using the service with predictable outcomes.

There is a regular structure in the *Query*, *Update* and *bulkLoad* operations. These operations will often take time to perform, as they may require access to or transferring of substantial amounts of data. We assume that each operation goes through three phases:

1. *Preparation and validation*, during which the operation is checked to ensure that it is internally consistent and that it conforms to the data model of the database.

2. *Application*, during which time updates are performed or the query evaluated, and results constructed in the form required as a result.

3. *Result Delivery*, during which time results are made available to the caller of the operation.

Since preparation and validation will normally be brief relative to the sum of the other phases, the operation returns after this phase. Errors at this stage normally indicate a logical failure. Immediate notification of such errors reduces the likelihood of a client continuing under false assumptions.

The final result of an operation is made available available later via the *resultHandle* parameter. This is the GSH of a structured data delivery system described in Section 4.3, providing asynchrony for long-running operations and an opportunity for redirection. Failures that take place during application of an operation are reported by way of the data delivery system.

The pairs *(queryNotation, query), (updateNotation, update) (loadNotation, valuesHandle)* and *(editNotation, edit)* are introduced to permit flexibility, in the same way that MIME types accommodate a variety of attachments with email. For example, any of the *queryNotation*, *updateNotation*, *loadNotation* and *editNotation* might be specified to be *SQL'92* and the corresponding second parameter would contain or yield a string in that notation. A single database service might often support several notations. For example, Xquery and Xpath might both be supported by an interface to an XML repository. When a database service registers with a discovery service, it must indicate which notations it is prepared to use. This information may periodically be modified during the lifetime of the service. A notation could be represented by a URI that refers to a definition of the notation.

The final results of an operation are managed via the triple *expires, resultHandle, resultDeliverer*. The parameter *expires* requests an expiry time up until which the result may be claimed. When omitted, a default value is used. As it requires resources to hold a result, this is necessary to recover those resources if the client fails to use them. The result handler is either generated dynamically

query(IN queryNotation, IN query, OPTIN values, OPTIN txHandle, OPTIN expires, OUT resultHandle, OUT fail)
update(IN updateNotation, IN update, OPTIN values, OPTIN txHandle, OPTIN expires, OUT resultHandle, OUT fail)
bulkLoad(IN loadNotation, IN valuesHandle, OPTIN txHandle, OPTIN expires, OUT resultHandle, OUT fail)
schemaUpdate(IN editNotation, IN edit, OPTIN txHandle, OUT fail)

Table 1: Operations supported by basic *database statement interface*.

or supplied. Mechanisms provided by the OGSA for managing the lifetimes of services are discussed in Section 5.2.

The operations in Table 1 have transaction handles (*txHandle*) as optional inputs. Where no transaction handle is given, assuming that the system implementing the database service supports transactions, each operation can be expected to run within, and constitute the complete behaviour of, a single transaction. A single transaction can be made to span multiple operations by passing the transaction's handle as a parameter. Support for transactions that span multiple statements or database services is discussed in Section 4.4.

4.3 Delivery Systems

A delivery system is a means by which (potentially large amounts of) structured data is moved from one location to one or more others. The delivery system mechanism should be considered complementary to protocols such as GridFTP, as the emphasis is on providing generic facilities for managing the internal structure of the data that is to be delivered. Thus a delivery system could, for example, use a GridFTP service as a delivery mechanism.

Table 2 shows a possible interface for a delivery system. A delivery system has a single source for the data to be delivered, represented as a URI, plus a collection of delivery mechanisms. Each delivery mechanism has a GSH for the service that will actually deliver the data and a *URI* describing where the data is to be delivered to. The *deliver* operation initiates the delivery of the data from the single source to the multiple destinations.

A more comprehensive deliver system could include facilities for conducting format mappings before or after delivery, encryption, progress monitoring, etc.

4.4 Transactions

Transactions are crucial to database management, in that they are central to reliability and concurrency control. Transactions, however, are not database-specific artifacts – other programs can and do provide transactional features through middleware services that conform to industry standards (e.g. OMG, J2EE). This section gives an indication of how a transaction service might be used in conjunction with a database service, and it also explains how such a service may be extended to satisfy a more flexible requirement to coordinate operations across a grid.

A minimal transaction interface that a grid service might support is given in Table 3. In this case, the interface is straightforward, and essentially performs the role of conferring a guaranteed unique identity on the transaction. It may additionally offer status enquiry and monitoring facilities. Given a transaction handle, other operations over a database service can be put explicitly within the context of a transaction, using the *txHandle* parameter introduced in Table 1.

The operations described to date allow a client to coordinate accesses to a single database service. However, it may be necessary for a transaction to span multiple services. If a database service is to be able to participate in a distributed transactions, it must provide operations for use by the transaction manager that is overseeing the distributed transaction. Such additional operations are illustrated in Table 4. The *startTransaction* operation has been extended to include an *expires* parameter to limit the consumption of resources – the transaction manager can request a duration for which the transaction will remain active, and the participating service may accept the request or return an alternative, probably reduced, period. The *prepareCommit* operation can be used by a two-phase

setSource(IN URI, OUT fail)
addChannel(IN deliveryMechanism, IN destURI, OPTIN expires, OUT fail)
deliver(OUT fail)

Table 2: Operations supported by basic *delivery-system interface*.

startTransaction(OUT txHandle , OUT fail)
rollback(IN txHandle, OUT fail)
commit(IN txHandle, OUT fail)

Table 3: Operations supported by a basic *transaction interface*.

startTransaction(OUT txHandle, OPTINOUT expires, OUT fail)
prepareCommit(IN txHandle, OPTIN expires, OUT fail)

Table 4: Operations required for participation in distributed transactions.

commit protocol to ensure that all or none of the database services participating in a distributed transaction commit.

Although transactions are a fundamental concept for database operations, the Grid requires additional and more flexible mechanisms for controlling requests and outcomes than are typically offered by traditional distributed and database transaction models. Some specific differences between the Grid and traditional object transaction environments are:

- Multi-site collaborations that often rely on asynchronous messaging. While this model also occurs in traditional distributed and database systems, the transactions in a traditional system are typically chained together rather than considered as a part of an overall concurrently executing collaboration.

- Operations across the Grid inherently are composed of business processes that span multiple regions of control. Such an environment contrasts significantly with traditional distributed and database systems, where the processing dedicates resources exclusively to the executing transaction (database locks, threads, and so on).

- Traditional distributed and database transaction models optimize execution for high-volume, short-duration transactions and conversations. Grid operations will typically be of longer duration.

Instead of simply extending an existing transaction model to the Grid, an incremental approach is suggested:

1. Construction of a core activity service model that provides the capability to specify an operational context for a request (or series of requests), controlling the duration of the activity, and defining the participants engaged in the outcome decision. An example of such a service is the Additional Structuring Mechanisms for the OTS Specification from the OMG [OMG00], which is also being adopted within the Java Activity Service.

2. Development of a High Level Service (HLS) that provides implementation of patterns typical in a Grid environment, e.g.

   - Traditional distributed and database model where operations occur completely or not at all. Such a completion processing semantic provides the behaviour of a traditional transaction model (i.e. a two-phase commitment semantic).
   - Relaxed semantics such as conversations or collaborations, where a series of operations

occur in a more loosely coordinated activity. The participants determine the requirements for completion processing, which may include patterns for compensation, reconciliation, or other styles of collaboration, as required.

Note that the requirement exists to provide a standardized client interface to allow applications to make use of any HLS implementation. A specific proposal for such a client API is outlined in Java, JSR000156 XML Transactioning API for Java (JAXTX), which is intended to provide a generic API supporting both transactions and extended transactions in a J2EE environment.

The behaviours cited by no means represent a complete list. The relaxed transaction model, however, allows the participating sites to supplement their existing implementations to support more complex processing and relaxed transaction processing models. The sites can build on their internal transaction and business logic environments to provide increased flexibility.

4.5 Database Metadata

The OGSA includes a standard, but abstract, discovery interface that all grid services should support. This existing interface provides the operations that can be used to obtain information about a database service. In essence, the discovery interface provides a query facility not unlike that described in Section 4.2 for requesting information about a service. As such, the questions left to the developers of a database service are: (i) what metadata should be provided to describe a database service; and (ii) how should this be modelled for convenient access by the discovery interface?

Database metadata that it could be useful to have access to includes:

Content description: The metamodel of the data in the database describes what the database contains. There are several categories of such data: the database schema – the data model, the logical and physical structures of the database, such as, respectively, the tables and indexes of a relational database; properties affecting access to and use of the data – authorisation, ownership, reliability and provenance data; statistical characteristics, such as the number of objects in a collection, the cardinality of a relationship, or the selectivity of an attribute. The statistical characteristics of a database are important, for example, for query optimization during distributed query processing, as described in Section 4.6.

Capability description: The capabilities of a DBMS are many and varied, and a service architecture must be able to accommodate systems with diverse facilities. This is supported by individual services making their capabilities known. There are several categories of such data, such as: language capabilities – which query and update operations are supported or made available; transactional capabilities – the transaction semantics supported by the system, including its facilities for participating in distributed transactions; connection options – the protocols and encodings that can be supported; and quality of service with respect to media and site failures.

The above metadata needs to be modelled in a manner that allows the discovery interface to be used to access the data. This could be done, for example, by allowing access to the data dictionary of a relational database using SQL. The data dictionary of a typical relational database stores much of the data described above. However, such an approach leaves the database metadata in a database service-specific format. It would be useful if there were standard representations of database metadata, for example as XML documents, provided by different database services. It would be the responsibility of the database service provider to describe the service in a standard way. This could involve supporting a mapping from stored metadata to the standard model, or manual specification of the model. However, significant portions of the metadata of a database are data model specific). In addition, a complete model for database metadata is likely to be complex – for example, the JDBC `DatabaseMetaData` interface includes over 150 methods. The identification of appropriate models for database metadata therefore requires significant further work.

As an aside, we discuss *data model variants*, to explain another category of metadata. Many data models are defined by standards, but these are not uniformly implemented. Often vendors introduce variations, or variations occur as a result of a succession of versions of a

standard. A service interface will normally make best efforts to avoid inflicting these variations on clients (JDBC [EHF01] also states this as one of its goals). However, it is sometimes necessary to consider these variations, e.g. when implementing code to hide them from clients. These variations should therefore be described and obtainable from some database services, although they should not concern most programmers.

4.6 Virtual Databases

The Grid is an environment for distributed computing. Although service architectures are designed for use within distributed environments, there has been no systematic attempt to indicate how techniques relating to distributed or federated database systems might be introduced into a Grid setting. This section provides some suggestions.

4.6.1 Distribution and Databases

Firstly, a few definitions. A *distributed database* is a database that has been deliberately distributed over a number of sites. For example, a gas exploration company may choose to store drilling information close to each of the exploration activities in which it is involved. A distributed database is designed as a whole, and is associated with considerable centralised control.

In a *federated database*, many databases contribute data and resources to a multi-database federation, but each participant has full local autonomy. In a *loosely-coupled* federated database, the schemas of the participating databases remain distinguishable, whereas in a *tightly-coupled* federated database, a global schema hides (to a greater or lesser extent) schematic and semantic differences between sources [MP00].

From the perspective of a service-based model, an existing distributed or federated database can be seen as a database service in which it happens that some or all of the data being accessed is remote from the system providing the service. As such, a distributed or federated database system provides standard database functionality, and could implement a database service using the interface described in Section 4.2. However, from the point of view of the service user, such an approach essentially encapsulates issues relating to distribution, and there could well be merit in developing virtual database services specifically for use in Grid settings. Such an activity stands to benefit from standard service definitions, as these provide uniform access to existing databases. A further opportunity exists to use Grid resource allocation facilities, etc, to support efficient distributed query evaluation.

4.6.2 Distributed Query Service

This section discusses how a Distributed Query Service could be developed, both providing and using database service descriptions. Assume that database services have been developed, supporting statement, transaction and metadata interfaces, as described above, and that multiple databases implement the services, and are available on the Grid. Any Grid application would be able to establish connections to multiple databases at the same time, and a distributed transaction service would allow coordination of queries and updates over the databases. However, any single query would only be able to refer to data from a single database, thereby requiring applications to subdivide and order requests that require access to data from more than one database. The role of a distributed query service is to allow individual queries to access multiple databases, thereby allowing the system to take responsibility for query optimization and efficient evaluation.

Figure 1 illustrates three database services, one of which (*Database Service 1*) happens to be a distributed query service, and two others, which we assume are not (i.e., they are directly associated with stored data). The administrator of *Database Service 1* has provided a schema for the distributed query processor to act over by importing schema information from *Database Service 2* and *Database Service 3*, as illustrated in Figure 1. It is assumed in Figure 1 that the participating databases and the distributed query service act over relations.

Figure 2 illustrates the distributed query processing service in action. Firstly, a query that joins data from tables in different databases is submitted to the distributed query service. The query is parsed and optimized, to produce an execution plan. Secondly, the plan is executed, which leads to subqueries being sent to the relevant databases – in this case, the tables required for the join are obtained from the relevant databases for joining by the distributed query processor. Thirdly, the results of the subqueries are collected and joined by the distributed query processor.

The technologies required to support such a distributed query service are not run-of-the-mill, but have been the subject of considerable research, and can be considered reasonably mature (e.g. [HKWY97]). However, these might benefit from some adaptation for incorporation into a Grid setting, for example, by making use of Grid resource management facilities to identify suitable computational resources for evaluating distributed queries or storing intermediate results.

A services architecture is useful to the development and deployment of a distributed query processor, as the "virtual database" that results from the use of the service: (i) can be advertised and accessed like any other database service; and (ii) can use consistent facilities to discover, integrate and query existing database services.

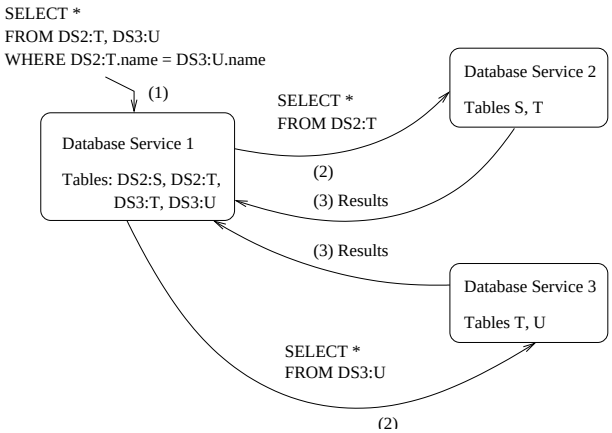

Figure 2: Using up a distributed query service.

4.6.3 Selective Replication

Replication functionalities for files are part of existing Grid middleware. Existing DBMSs also provide replication capabilities. For example, a relational DBMS may be able to replicate some or all of a relation, and to provide alternative update policies. One such update policy could involve one copy of a table being available for read or update, and the others available only for update. An alternative may involve read and write access to any of the copies of the table, with either strict concurrency control or some form of conflict resolution.

It is reasonable to assume that selective replication capabilities could be useful in e-Science applications. Such facilities could, for example, extend to the use of remote view materialisation, with incremental propagation of updates. In such a scheme, the information to be replicated is described as a named database query (a view). This view is then materialized at some location. A notification service can then be used to detect updates that should be reflected in the materialised view, so that relevant changes to the view are propagated to the view automatically.

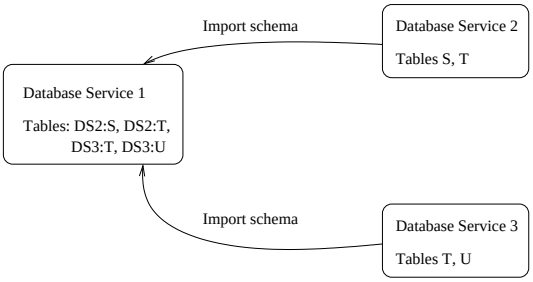

Figure 1: Setting up a distributed query service.

As with other functionality of an existing database system, replication mechanisms within existing database systems could be wrapped and made visible through the generic services interface. This would require few if any changes to the query and update interfaces, but the presence of replicated data could usefully be made explicit through the metadata service. Knowledge about DBMS-maintained replicated data could be used by Grid scheduling services.

Providing additional replication services within the Grid setting, for example through an incremental view maintenance scheme based on a distributed query processing service (as discussed in Section 4.6.2), would require the development of significant extensions to the distributed query processor. However, it would allow significant additional functionalities compared with those currently available (which are intra-vendor), could provide significant added-value for a distributed query service, and could fit in well with current practices for the storage and analysis of scientific data.

4.7 Connection Services

The general architecture for database services described above is more complex to use than established connection APIs such as JDBC and ODBC. This is unsurprising, since those protocols eliminate most of the flexibility needed for grid applications. For example, they fix the data model and parameter representations, they eliminate the possibility of obtaining values from or sending values to third parties, and they use synchronous operations. There are two significant reasons for doing this:

1. They are simpler to understand and use, and application programmers benefit from that simplicity when the generality and flexibility is not needed.

2. It permits optimisations, such as caching information about the database at the client end, caching and buffering results, data flow management optimised for the particular pair of end points, private value representations, etc.

In the short term particularly, the first factor will be important in encouraging projects currently using traditional DBMS connections to adopt the service-based approach. A connection service should therefore be considered as the logical equivalent of consistently partially applying the operations of a particular GDS with standard values for such parameters as: *updateNotation*, *expires*, *resultDeliverer*, *editNotation*, *queryNotation*, etc.

Such a service should be straightforward to use and provide good average-case performance. In addition, it would be possible to cache at the client the content metadata of the target schema, and validate operations before transmitting them. This has the major advantage of enabling prompt detection of many errors without communication costs and delays. It would also convert the asynchronous API into a synchronous API by obtaining result sets before returning.

4.8 Management Services

There are clearly management tasks associated both with databases and with a Grid environment. Thus services could be envisaged specifically for creating, administering, monitoring and maintaining databases within a Grid setting. Such services might be considered higher-level than those described above, but are nevertheless important to containing the cost of database activities within the Grid. The publication of information about a database service so that it can subsequently be identified and used can also be seen as a necessary management service. The development of management facilities for database services involves relationships both with the management services of existing databases and with wider Grid management services.

5 Database Services in OGSA

The interfaces described in Section 4 have been presented with minimal reference to a broader services architecture. Thus many features of an architectural framework, such as how services are created, evolved and destroyed, how access to services is controlled, and how notifications are passed between services, have not so far been discussed. This section discusses how the interfaces described in Section 4 can be used as the basis for database services that obtain these additional characteristics through participation in the OGSA.

5.1 Service Granularity

The interfaces described in Section 4 can be imported by service descriptions. As such, specific services may import different interfaces, and thus support different functionalities. The following would provide coherent capabilities:

BasicDatabaseService: such a service would import the interfaces from Table 1. Such a service would provide minimal transaction facilities, as each operation invoked on the service would be run independently of all others.

TransactionalDatabaseService: such a service description would import the interfaces in Tables 1 and 3. Such a service would allow transactions to span multiple query and update operations, but would not be able to participate in distributed transactions using two-phase commit.

CoordinatableDatabaseService: such a service description would import the interfaces in Tables 1, 3 and 4.

Such a service would allow transactions to span multiple query and update operations, and would be able to participate in distributed transactions using two-phase commit.

Other (non-database) services would be able to include transactional behaviour in a consistent manner by importing (and implementing) the operations from Tables 3 and 4. It is assumed that a delivery system, an interface for which was described in Section 4.3, is a distinct service.

5.2 Life Cycles

The OGSA distinguishes between *persistent* and *transient* services. A *persistent* service is created outside the grid service environment, yet made available for discovery, access, etc, using standard grid service conventions. By contrast, a *transient* grid service is typically created within the OGSA. A transient service is created by a *factory*, for which a standard interface is defined. This interface allows information on such properties of a transient service as its lifespan and access rights to be specified when it is created. In the case of a database service factory, additional information such as the schema of the database might be given when the transient service is created.

A GDS can be a transient or persistent grid service. In the former case, the GDS is established within OGSA and operates entirely within the OGSA framework. Though it is transient, it may be long-lived. In the latter case, the creation, management and possible termination of the GDS is beyond the realm of OGSA.

Where a database system is managed outside the OGSA, it may not be accessible directly via OGSA protocols because it doesn't yet support them. In this case, a proxy behaving as a a GDS can permit standard OGSA behaviour and bridge this to the established DBMS. The proxy could be transient or persistent, and therefore itself potentially subject to OGSA life cycle management. We hope that as the OGSA develops, database management systems will directly support the protocols and APIs of the OGSA, as this will improve efficiency and lead to beneficial co-evolution.

Operations for managing the lifetime of a transient service are specified as part of OGSA. These allow explicit or soft-state description [FKNT02].

As a database factory is itself a service, it must be able to describe its capabilities. If an application is considering creating a GDS, it must find a database factory service (DFS) that is capable of producing a GDS of the right form. It is no use asking a factory specialising in creating services that perform similarity searches over protein sequences to generate a GDS for managing XML data according to a specific XML schema. Consequently, when registering with a discovery service, the DFS must specify the classes of GDS it is able to create and manage. The capabilities of the services that can be created should be described using representations that are consistent with the descriptions of services that already have been created, as discussed in Section 4.5.

5.3 Types and Namespaces

Types used in the definition of database service operations need to be represented in the XML Schema namespace [TBMM01] used in the WSDL definitions of database services. This namespace should have a standard URI, such as `http://schemas.ogsa/GDS/parameters/`. This should cover the types used for describing results, status, errors, etc.

For each data model there will need to be types defined for transmitting parameter values and result sets. These need to use common namespaces at URIs with a general form such as `http://schemas.ogsa/GDS/<data-model>.values/`. In addition, many scientific applications that are prevalent users of the Grid have commonly occurring types that will be used in queries and results. For example, complex numbers represented with a defined numerical precision, matrices, banded matrices, diagonal matrices, etc. To prevent unnecessary translations and redefinitions a namespace containing a set of types for these should be established at URI such as `http://schemas.ogsa/scientific/data/`.

5.4 Authorization

The OGSA assumes that authentication of a requesting party is carried out by the network protocol binding. This results in the service being able to identify the authenticated requestor by way of a URI. In addition, an interface

is provided that allows it to be stated who is able to invoke each operation under what conditions.

However, in database systems, authorization is often sophisticated, and fine grained control can be provided over access to or update of data items. Furthermore, authorization facilities may check that the person on behalf of whom an update is requested is working in a group identified in the data to be updated. Overall, a wide range of authorization constraints may be encountered in database system; it is beyond the scope of this document to explore the issues associated with making database authorization facilities fully accessible within the OGSA.

5.5 Notification

Notification services allow asynchronous communication of events and event descriptions. In the OGSA, interface definitions are provided what allow services to act as notification sources or sinks.

A GDS should be able to act as a source of notifications, as other Grid services may be interested to know, for example, of changes made to a database or operations taking place within a database. Further, it is also clear that a database may seek to receive and respond to notifications – for example, a database supporting an Information Service may need to be updated to reflect changes in the status of another Grid entity.

Some database systems come with reactive capabilities; for example, the SQL-99 trigger facility is described in [KMC99]. It is not always the case, however, that such facilities are able to monitor events taking place outside the database itself, or to invoke operations that are immediately visible outside the database. It is straightforward to use a database wrapper to forward external notifications to a database, but more challenging to cause a passive database to inform interested external parties of relevant events.

6 Examples of Grid Database Services

To explore the Grid Database Services approach, we consider a few illustrative examples. In so doing, we don't cover relational or object databases directly, as it is hopefully fairly obvious how they fit within the framework.

6.1 An XML Repository

Consider a simple GDS that handles concurrent queries over an XML repository, but for which updates require exclusive access to the service. Such a concurrency control scheme could be implemented using a file system's locking mechanisms. As discussed in Section 5.2, if the service is to be transient, an *XML Factory Service* will be required that can create instances of the *XML Repository Service*, with the service creation operation taking various parameters relating to the lifespan, access rights and schema of the service.

The operations supported by the service are essentially those in Table 1. However, the straightforward XML repository only supports one form of *updateNotation* that loads an XML document, and verifies that it complies with the stored XML schema. If it complies, it stores the document, otherwise it records the errors in *result* and reports a failure in *fail*. This simple service might support only one *queryNotation*, such as XPath, and may only be able to deliver results using a limited range of protocols.

More advanced versions of the service could extend its capabilities without requiring a change to the signature of the operations. For example:

1. An *updateNotation* called *insertXML* might specify a path to a location in the *update* parameter, where the XML document specified by the *values* parameter is to be inserted. The resulting document would have to comply with the bound schema or the update would fail.

2. There might be an index to accelerate queries.

3. All versions of the document's history might be retrievable.

4. The query operation might accept a *queryNotation* of either XPath or Xquery, with an optional time parameter. Specifying a time would cause the evaluation to occur as if it was against the document at the state it was in at that time.

5. The storage regime might store quiescent documents in compressed form.

This range of extensions illustrates that there are many variations and developments possible within the services

framework when considering the provision of just one service. The actual operations and development for a particular service would be driven by application requirements.

6.2 A Sequence Repository

This section describes a biological *Sequence Repository Service* as a composition of two existing services. One of these services is the *XML Repository Service* from Section 6.1. The other is a *Sequence Search Service* that allows similarity searching on sequences. It is assumed that an application needs to associate some application metadata encoded in XML (for example, on experimental methods) with the actual sequence data. By combining the capabilities of the *XML Repository Service* for storing the metadata and the *Sequence Search Service* for conducting similarity searches, a new *Sequence Repository Service* can be developed.

Two new services must be developed:

Sequence Repository Factory Service: the creation of a sequence repository service requires the creation of an *XML Repository Service* and a *Sequence Search Service*. As such, the *create* operation of the *Sequence Repository Factory Service* must call the two factory services that are able to create the needed *XML Repository* and *Sequence Search* services, the GHSs of which are stored by the newly created *Sequence Repository Service*.

Sequence Repository Service: the operations of this service are essentially those from Table 1. The *query* operation might support a number of functions that combine searches over the XML and sequence repositories. For example, a search could retrieve the XML metadata of all sequences that are similar to a given sequence.

This kind of application requirement, where experimental results (sequences in the example) are paired with some additional information about the way the data was collected or some manual annotations describing the data, is quite common in e-Science. The example has shown how the GDS service interface can be used to package and combine rather diverse capabilities.

6.3 A Matrix Storage Service

This section assumes that there are enough applications that wish to store matrices to make a specialised service useful. Indeed, there are already a variety of matrix storage schemes in use in Grid projects. Assuming that a *Matrix Storage Service* may be transient, we simply introduce a *Matrix Factory Service* and a *Matrix Storage Service*.

The developers of such a service within a grid database service context have a range of design issues to face. For example:

1. Is it worth developing a standard notation for describing matrices to be stored, call it a Matrix Description Language (MDL)?

2. Is it worth developing a standard notation for extracting data from matrices, call it a Matrix Query Language (MQL).

The actual form and detailed design of such languages will not be considered here, only the rationale for designing and supporting them. Consider first an MDL. Notations such as $\tau[l_1, u_1; l_2, u_2; \ldots; l_n, u_n]$ are familiar as a description of the shape of an n-dimensional hyper-cuboid array of elements of type τ. It is possible to extend this notation to describe other forms of (constraints on) matrices, e.g. symmetric, triangular, or to indicate information useful to storage systems, e.g. sparseness.

As suggested, the description may be used both to organise validation of operations, and to optimise storage and operation evaluation. In addition, it may be used as information to clients about the properties of the data. This is achieved at the cost of complexity in implementing the service. The trade-off would need investigation.

Once the structure of a matrix store is defined, it is possible to introduce specialised query languages that use this structure. DataCutter [BFK+00] is a particular example. Such languages achieve an important optimisation, moving selection close to data, rather than moving large volumes of data to a process. But DataCutter essentially provides a language for identifying hyper-cuboid slices of a matrix and filtering those. There are many other selections possible. More importantly, it is often some derivative of a subset of the matrix that is required, e.g. a random sample, an average, a standard deviation,

a fitted hyper-surface and residuals, etc. This suggests that attempts to capture the full gamut of requirements in a special purpose matrix query language may be futile. However, frequently occurring requirements might be met by a language, whereas general requirements can only be met by using a general purpose programming language. For safety, this either requires that the data is moved to a client's process where this code is executed, or that the GMSS provide a mechanism for accepting a "query" as code, e.g. a Java compiled class satisfying a specified signature, dynamically binding this into a sand-boxed context and running it against the stored matrices.

7 Developing Grid Database Services

We consider here two aspects of the further development of Grid Database Services: the requirements it imposes on OGSA and the issues in the OGSA model that require development.

7.1 Requirements from OGSA

Many of the planned features of OGSA will be exploited by grid database services. A few examples illustrate the synergy.

1. All activations and parameter/result transmission depend on the standard OGSA mechanisms.
2. The secure connection and authentication mechanism underpins all GDS security and authentication.
3. The lifetime management model carries over unchanged as the lifetime management model for GDS.
4. The soft-state model applies to all of the transient service instances, such as connections, delivery services and transactions.
5. The notification mechanism specified in OGSA appears to satisfy the GDS needs.

It is instructive to try to identify where extensions to the current GS infrastructure are likely to be necessary. This will only be clarified by exploratory prototypes that test the capacity of OGSA to carry GDS implementation, and the capacity of DBMS to conform to GS requirements. Some anticipated issues are:

1. Supporting the authorisation requirements of external databases. The database will require information about the user causing the activation, potentially through many intermediate grid services. A certificate establishes a bone fide identity, but yields little information about the properties of the user that may be needed to establish entitlement to perform an operation. A way forward that retains flexibility could be to introduce user identification services, and references to them in a certificate.
2. Certification of the services themselves may be necessary. The trust needed to commit data to some service and repository has to be established. There is a risk that a discovery service has been tricked into returning a GDS that mimics the intended GDS but exposes the data sent or the area being probed. Does this require digital signatures for GS that can be verified by a validation authority?
3. The types needed for e-Science applications and for database integration need to be defined as XML Schema namespaces. If this is not done, different e-Science application groups will develop their own standards, which will generate integration problems and eliminate potential economies.
4. Some external databases charge for their use. To allow these to be used via the grid it is necessary to support a digital payment process. Since payment is in arrears, and since a user may be spending concurrently via many grid services, a sophisticated, distributed credit rating/accounting system must be available.
5. Extensions of information discovery services may be required to handle extra properties, e.g. transactional and coordinatable. An extension will also be needed to handle content information, and the structures for describing content will need to be agreed.

7.2 Issues for Grid Database Services

Conversely, the provision of grid database services places demands on the service providers. Some examples illustrate the issues that arise.

1. External persistent database systems must allow connection from the grid. The associated technical problems can be overcome, but the political challenge of persuading the owners of the data to permit access to a large number of remote users via the grid may inhibit development.

2. When a GDS is used as a transient service, installation and set-up becomes an issue. Present systems require system administrators and database administrators to perform incantations establishing operational parameters and context before operations can commence. An alternative is required, driven via the services infrastructure, using automatically derived information about the context and parameters.

3. A database system on which a GDS is based could be vulnerable to accidental or deliberate overload. Mechanisms must therefore be developed that enable the resources used by each client to be appropriately controlled or scheduled.

4. In the grid context, it is necessary to perform co-scheduling and reservation, so that other resources are not squandered because they are waiting for a resource that is not yet available. Interfaces for such scheduling and reservation will need to be developed. Some DBMSs provide scheduled production of a snapshot, essentially of evaluation of a previously defined query, but none at present will reserve resources to evaluate a query or accept an update at some specified future moment.

5. DBMSs often require careful tuning to handle their expected workload. The sharing and distribution inherent in service-based computation may lead to unpredictable and less stable loads. The DBMS will need to respond to this with automatic tuning. In the interim, the operations for monitoring and tuning will need to be accessible via the service interface.

6. Best performance will be achieved by establishing infrastructure and computation close to data. For example, with large bodies of scientific data, performance can be substantially improved by using indexes that are tuned to current requirements [HAI01, SKT+00]. These are often large. They would need to be dynamically created and installed as new usage emerged. Where data has to be scanned, it is normally best to move the scanning algorithm to the data. This requires a safe mechanism for dynamically installing it close to the data.

7.3 Co-evolving Data and Grid Services

The preceding sections suggest that there are likely to be changes to both the grid service mechanisms and the data management mechanisms as a result of developing GDS. This is happening in an era of significant development in the treatment of scientific data as new approaches to archiving, provenance, integration and interpretation are emerging. Tools that derive and use semantic models to assist with these processes will emerge [RJS02] and substantially change the patterns of use and consequent workloads. There are also dramatic changes in the scale of data. For example, astronomy data collection rates are doubling every year, the available biological sequence data are doubling every 9 months and there is a marked trend towards digital data capture in medical imaging. These forces will not abate in the foreseeable future. Similar effects pervade e-business. The development of a robust framework for grid data services is therefore of great importance, as our ability to adapt and develop will be determined by the quality of its design. The territory must be explored through prototypes until sufficient understanding is available to guide this design.

8 Incremental Development

This section makes some tentative suggestions as to how a staged development of the services described in this document might proceed.

1. *Source-at-a-time, Single Paradigm.* The most straightforward database service would allow direct connections to individual sources, where these sources belong to a single database paradigm. The most obvious initial paradigm would be relational.

An initial implementation phase would be expected to provide no inter-service transactions.

2. *Source-at-a-time, Multiple Paradigm.* A good early test of the generality of the overall service specification would be to develop support for a second database paradigm, for example, for object databases or XML repositories.

3. *Multiple source, Loose Coupling.* Coordinated access to multiple sources could be supported by developing or deploying a distributed transaction manager. This would allow multiple connections to participate in a single transaction, but would not allow a single query or update statement to refer to more than one source.

4. *Multiple source, Tight Coupling.* This level of functionality would allow a single query or update statement to refer to more than one source, and thus involves the development of a distributed query processor. A distributed query processor could be developed before a distributed transaction manager, but this would reduce the coherence of the results of distributed queries.

5. *Multiple source, Tight Coupling with Replication.* Comprehensive and fine grained replication facilities build on both distributed query processing and distributed transaction models. More coarse grained replication schemes do not need a distributed query processor.

9 Conclusions

This document has made a preliminary, service-oriented, proposal for integrating database functionality into a Grid setting. It is hoped that the document will provoke discussion on how best databases can be integrated with Grid middleware, and subsequently that descendents of this document can direct implementation efforts relating to databases and the Grid.

Please feel invited to provide feedback, and to suggest improvements, preferably be email to: norm@cs.man.ac.uk.

Acknowledgements: The authors are grateful to Ian Foster and Heinz Stockinger for their feedback on earlier versions of the document.

References

[BFK+00] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proc. 9th Heterogeneous Computing Workshop (HCW)*, 2000. Cancun, Mexico.

[BMRW98] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In *Proc. CASCON*, pages 4–18, 1998.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C Note, www.w3.org/TR/wsdl, 2001.

[EHF01] J. Ellis, L. Ho, and M. Fisher. Jdbc 3.0 specification. Technical Report Proposed Final Draft 3, Sun Microsystems, 2001.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Open Grid Services Architecture: A Unifying Framework for Distributed Systems Integration. Technical report, Globus Project Technical Report, www.globus.org/research/papers/ogsa.pdf, 2002.

[H+01] W. Hoschek et al. Data management (wp2) architecture report. Technical Report DataGrid-02-D2.2-0103-1_2, European Data Grid, 2001.

[HAI01] E. Hunt, M.P. Atkinson, and R.W. Irving. A database index to large biological sequences. In *Proc. 27th VLDB*, pages 139–148. Morgan-Kaufmann, 2001.

[HKWY97] L. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proc. VLDB*, pages 276–285. Morgan-Kaufmann, 1997.

[KMC99] K. Kulkarni, N. Mattos, and R. Cochrane. Active Database Features in SQL3. In N.W. Paton, editor, *Active Rules in Database Systems*, pages 197–291. Springer-Verlag, 1999.

[Kre01] H. Kreger. Web Services Conceptual Architecture. Technical Report WCSA 1.0, IBM Software Group, 2001.

[MP00] P. McBrien and A. Poulovassilis. Distributed Databases. In M. Piattini and O. Diaz, editors, *Advanced Database Technology and Design*, pages 291–327. Artech House, 2000.

[OMG00] OMG. Additional structuring mechanisms for the ots specification. Technical Report ORBOS/2000-04-02, Object Management Group, 2000.

[Pea02] D. Pearson. Grid database requirements. Technical Report http://www.cs.man.ac.uk/grid-db/, Paper for Databases and the Grid BOF, GGF4, 2002.

[RJS02] D. De Roure, N. Jennings, and N. Shadbolt. The semantic grid. Technical report, Southampton University, http://www.semanticgrid.org/, 2002.

[SKT+00] A. Szalay, P. Z. Kunszt, A. Thakar, J. Gray, and D. R. Slut. Designing and mining multiterabyte astronomy archives: The sloan digital sky survey. In *Proc. ACM SIGMOD*, pages 451–462. ACM Press, 2000.

[TBMM01] H.S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. Technical report, W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, 2001.

[Wat01] P. Watson. Databases and the Grid. Technical Report CS-TR-755, University of Newcastle, 2001.