Delivering Data Management for Engineers on the Grid

Jasmin Wason, Marc Molinari, Zhuoan Jiao & Simon Cox

School of Engineering Sciences, University of Southampton, UK
{j.l.wason, m.molinari, z.jiao, sjc}@soton.ac.uk

Abstract. Engineering design search and optimisation is a computationally expensive and data intensive process. Grid technology offers great potential to increase the efficiency of this process and many on-going activities focus on utilising computing resources on the grid. However, it is equally important to manage efficiently the vast amount of data produced by grid applications, so that data can be shared and reused. In this paper we describe the design and implementation of a database toolkit for engineers, which has been incorporated into the Matlab environment, to help them manage the large amount of data created in distributed applications. In particular the toolkit is built using secure file transfer (GSI, GridFTP) and open standards exchange of XML metadata between heterogeneous Web services, databases and platform independent Java clients. We show an application exemplar of how this toolkit may be used in a grid-enabled Computational Electromagnetics design search.

1. Introduction

Engineering design search and optimization (EDSO) is the process whereby engineering modelling and analysis are exploited to yield improved designs. Design parameters that an engineer wishes to optimise are identified, and the optimisation problem is specified by design variables, a design objective (objective function), and some constraints. The objective function which measures the quality of a particular design is computed using an appropriate model. The optimizing algorithm, which serves as the design modifier, perturbs each of a selected set of design variables to determine how they affect the performance. It is coupled with an appropriate engineering analysis code, such as Computational Electromagnetics (CEM) or Computational Fluid Dynamics (CFD) code, to analyse the properties of a design, and seek a solution that optimizes the objective. This process may involve lengthy and repetitive calculations requiring access to significant computational resources.

Grid technology [1] enables large-scale resource sharing and coordinated problem solving within a virtual organisation (VO) - a collection of geographically distributed and loosely coordinated groups. By providing scalable, secure, high-performance mechanisms for discovering, accessing and utilizing remote resources, Grid technology makes scientific collaborations achievable in ways that were previously impossible. The Grid allows the sharing of computing power, data resources and software applications over the Internet.

Requirements for long running, compute intensive calculations make the problem domain of engineering design search and optimisation using CEM/CFD well-suited to

www.manaraa.com

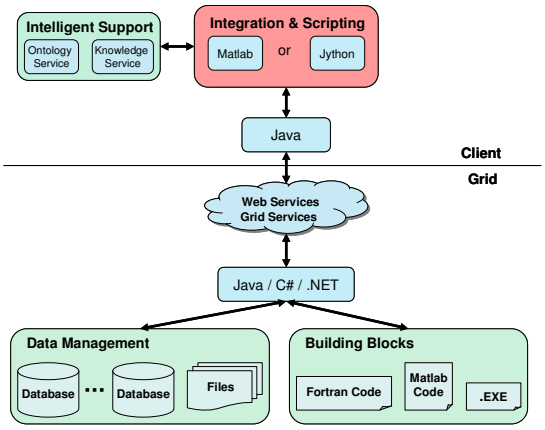

Figure 1 The Geodise architecture, where a scripting environment (e.g. Matlab) can be used to integrate grid-based compute, data and knowledge resources for engineering applications.

the applications of Grid technology. In Geodise [2] our goal is to develop sophisticated but easy to use toolkits to help engineers make use of the resources available on the Grid. We aim to provide data management and knowledge advice services [3] to help users to setup, modify and optimise their designs efficiently.

While compute power may be easily linked into applications using grid computing technologies, there has been less focus on database integration, and even less still on providing it in an environment familiar to engineers. In EDSO databases play a critical role as a repository for the results from the large number of calculations performed, which may be re-examined and reprocessed as part of the EDSO process.

Traditionally, data in many scientific and engineering disciplines have been organized in application-specific file structures, and a great deal of data accessed within current Grid environments still exists in this form [4]. When there are a large number of files it becomes difficult to find, compare and share the data. We solve this problem by using database technologies to store additional information (called metadata) describing the nature of the files, so that they can be located easily by querying the metadata. Access authorisation information is also stored in the database to allow users to setup permissions for data sharing.

The Storage Resource Broker (SRB) [5] is a client-server middleware addressing the issue of providing a uniform interface for connecting to heterogeneous data resources over a network. Its Metadata Catalog (MCAT) allows access to data sets based on their attributes (metadata) rather than their names or physical locations. However, MCAT has limited support for application specific metadata which is often essential in assisting engineering users to locate data specific to their problems.

We adopt open standards (e.g. XML) and a service oriented approach [6] to leverage existing database technologies and make them accessible in environments that engineers use daily. We have chosen Matlab [7], which is popular in engineering community for its ease of use and rich functionality, as such an environment. We have developed a set of tools to extend the Matlab functionalities so that engineers can make use of the grid computing and data management service in a way that is familiar to them. These consist of the Geodise computational toolkit [8], XML

www.manaraa.com

toolbox [10] and a database toolkit which is the focus of this paper. These toolkits can easily be adapted to work in other scripting environments, such as Jython [11].

As shown in Figure 1, the building blocks of the engineering software and the data management functionality provided by Geodise are wrapped as Web/Grid services when appropriate, making use of Java, Grid and .NET technologies. A user accesses these services via Geodise provided functions which in turn call a client side Java API to communicate with the services. The knowledge service, as described in [3], provides intelligent support to the users through an ontology service and dynamic advice based on data stored in a database.

The remainder of this paper is organized as follows: section 2 describes the architecture of the Geodise database toolkit; section 3 explains how the service components have been incorporated into an engineer-friendly Problem Solving Environment (PSE); an example is given in section 4, and section 5 presents the conclusion and future work.

2. Architecture

A major aim of the Geodise data management architecture is to bring together flexible, modular components for managing data on the Grid which can be utilized by higher level applications. Another objective is to provide a simple, transparent way for engineering users in a VO to archive files in a repository along with additional metadata. A simulation or optimisation may take a long time and it may be desirable to store files in a database for later re-use/ re-analysis to avoid costly re-computation. Although files can be archived and retrieved based on unique identifiers, storing additional metadata makes it easier to find a particular file by performing queries over more meaningful features, like the design job specification data, creation date, or values of design parameters. The user should be able to specify who else can discover and retrieve their data and be able to query data they have access to in the VO, without needing to know the underlying storage mechanism.

Such an environment for data management has been provided by the Geodise database toolkit, by using open standard technologies from an engineering environment. Various types of technical and application specific metadata about files, their locations and access rights are stored in databases. Files are stored in file systems and transported using the Globus Toolkit [9] which provides middleware for building computational grids and their applications. Commodity Grid (CoG) Kits [13] expose these functionalities as APIs in languages such as Java, Python and CORBA. We use the platform independent Java CoG kit to utilize the Grid Security Infrastructure (GSI) [14] for authentication and security, and GridFTP [15] for secure file transfer. As shown in Figure 2**,** client side tools initiate file transfer and call Web services for metadata storage, query, authorisation and file location. We now give further details of these components of the Geodise database toolkit.

Accessibility of data by users from remote sites in a VO should be considered to enable collaboration. To accommodate this we use the secure and reliable GridFTP protocol to transfer files to storage servers, archiving them under a specifically generated UUID (Universally Unique IDentifier) [16] as a file handle.

Access to databases is provided through Web services [6], self-describing programmable components that can be invoked over the Web. Web service methods

www.manaraa.com

may be invoked using the Simple Object Access Protocol (SOAP) [12], which uses a combination of XML and HTTP to transfer data between the services regardless of their underlying programming language or platform. For example, our Java client code running on Linux or Windows can communicate with .NET Web services on a Windows server and Java Web services on a Linux server. Client jobs running remotely on the Grid can also access these services to retrieve input files from and archive results to a repository. This provides a central location for applications running on the Grid to exchange data and allows the user to query and retrieve job results when convenient.

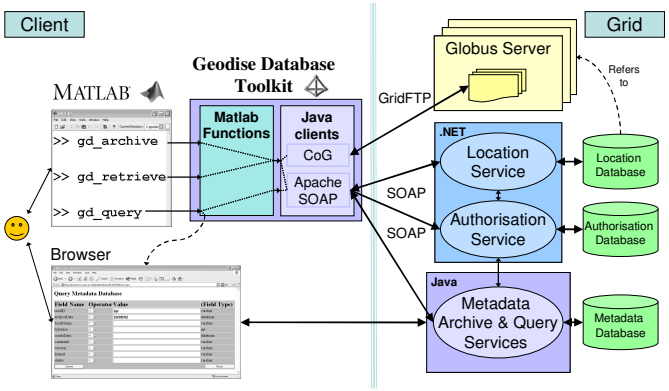

Figure 2 A high level set of scripting functions sits on top of a client side Java API to provide an interface to data management Web service functionality and secure file transfer. These may run locally or on a remote compute resource with the user's authorisation credentials.

The *file location service* keeps a record of file handles and locations, in terms of host and directory, in a database so that a handle is all that is required to locate and retrieve a file. The *metadata archive service* complements file archiving by allowing the storage of additional descriptive information detailing a combination of technical characteristics (e.g. size, format) and application domain specific metadata. The *metadata query service* provides a facility for engineers to find the data required without the need to remember the file names and handles.

Authorisation is implemented as a service interface to a database of registered users, keeping track of permissions on data and mapping between user IDs in the VO and Globus Distinguished Names (DN) which are globally unique identifiers representing individuals. The *authorisation service* filters query results and only returns metadata about files that the user has access rights to. Authentication is achieved with GSI which uses Public Key Infrastructure (PKI) [18] and Secure Sockets Layer (SSL) [19] to provide secure communication over the Grid. Every user must have a private key and a certificate, containing their DN and public key, which is signed by a Certificate Authority.

We use relational databases [22] for structured data such as authorisation information because they are mature, reliable and scalable, and have a well defined standard interface which allows generic tools to be developed for operations such as creating interfaces and constructing queries. We also use XML repositories for more flexible storage of complex, deeply nested engineering specific metadata. In this version we use the XML native database Xindice 1.0 [19] which has proved a very

www.manaraa.com

flexible solution for prototyping management of engineering metadata. However, the current version does not provide the levels of scalability and security we require. Our requirements for flexibility and robustness may be met by a relational database system with XML capabilities, for example Oracle9i Database [20], IBM DB2 [21], and Microsoft SQL Server 2000 [22]. We require a set of services that allow us to access and interrogate both types of data storage in a standard way. Other projects are tackling this problem for relational databases [23] and XML repositories [24]. We currently provide APIs to specific databases with tailored Web services and will use these on top of implementations compliant with proposed standards from the OGSA [1] Data Access and Integration project [4] and the GGF [25].

3. Problem Solving Environment

Matlab is a powerful scripting environment containing a large number of toolboxes tailored to the needs of scientists and engineers. It is a PSE in which users can learn quickly how to generate, analyse and visualize their data. Its database toolbox [26] uses JDBC to enable insertion and retrieval of data between Matlab and relational databases. This is a useful tool for users who already have a local relational database they wish to access from Matlab. However, this technology is inappropriate for engineers who do not have an existing database, and are not concerned with the underlying storage schema (e.g. tables and columns) used to store their data. We have implemented a range of grid-enabled Matlab functions (`gd_archive`, `gd_retrieve`, `gd_query`) on top of our core database services so that they can be incorporated programmatically into the user's scripts in a way that is consistent with the behaviour and syntax of the Matlab environment. The basic tasks for an engineer to undertake to manage and share their data are to (A) *generate the data* using their standard engineering tools, (B) *store it* in the repository, (C) *search for data* of interest, and (D) *retrieve results* to their local file system. The wrapping of the core services that enable these tasks is straightforward as much of the logic of the client side components is written in Java, which can be directly exposed to Matlab or other high-level scripting environments. For example, it has been proven straightforward to write wrappers to expose the client side functionality of the Geodise computational toolkit to Jython [11], a pure Java implementation of the Python interpretive scripting environment.

The `gd_archive` function stores a given file in a repository for an authenticated user. The function is able to generate standard metadata for the file, such as its local name, size and format. The user may add additional metadata, for example custom application specific information, and a list of users who may access the file. The function then transports the file to a server and the metadata to the metadata service for storage. The `gd_archive` function returns a unique handle which can be used to retrieve the file at a later date. The locations of databases and file servers are set in a configuration file by an administrator of the system. The `gd_retrieve` function will locate a file based on a given file handle and return it to a local directory.

Matlab allows users to define complex structures in a straightforward way, and this is how custom metadata is specified. We provide capabilities for direct conversion from these structures to XML, for storage in the metadata database. The metadata can be queried by an authorised user with the `gd_query` command, to discover files that have certain characteristics and obtain information about them, such as their handles.

www.manaraa.com

Users specify the queries in their scripts using a combination of named metadata variables and comparison operators. For example, *'standard.archiveDate>2003-02-01 & param.radius = 2.3'*. An alternative graphical query interface is also provided in which selection criteria are specified in a Web form generated from standard metadata (Figure 2, bottom left). In this interface there is an option to generate the Matlab code needed for retrieving selected files that match the search criteria, which can then be cut-and-pasted into the user's own Matlab script.

When a script based query is performed, the XML metadata results are converted back into a Matlab structure which is returned to the PSE. The bi-directional conversion routines are provided by our XML Toolbox for Matlab [10]. The toolbox includes functions to convert Matlab variables into XML and vice versa. We use the XML toolbox as an underlying tool that the user does not see when calling our database functions. As far as the user is concerned they are archiving, querying and working with Matlab structures, not XML.

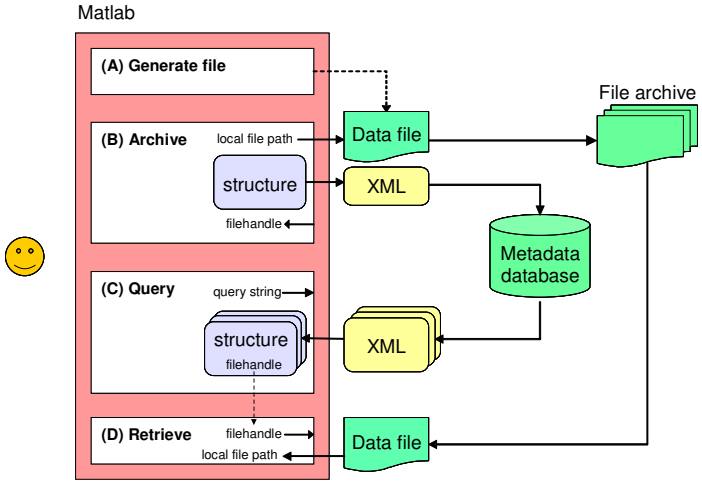

Figure 3 Data flow of files and metadata. After file generation (A), the user specifies metadata and archives it together with the file (B). When querying (C), a file handle is returned as part of the metadata structure and the file can then be retrieved to a local location (D).

Figure 3 shows the Geodise database toolkit from a dataflow perspective. A more detailed description is given in next section, to show how the Geodise database toolkit has been used in a real world example of CEM design search.

4. Application Example

Let us now consider how databases may be used in a Grid environment for a specific example: The GEM (Grid-enabled electromagnetic optimisation) project [27] is developing software which is used to improve the design of optical components for next generation integrated photonic devices.

Suppose an engineer wants to optimize the electromagnetic transmission properties of such an optical component, for example a photonic crystal (PC). A PC consists of a periodic structure of holes drilled into a slab of dielectric material as shown in Figure

www.manaraa.com

4. The properties and density of the holes determine the transmission properties of light through the crystal. There exist certain designs which exhibit a reduction of light propagation within a specific frequency range, called a photonic bandgap (PBG) of the crystal. It is often desirable to obtain a large bandgap in the frequency spectrum. The size of the bandgap for a number of geometries can be investigated by sampling a range of parameters (in this case the radius *r* and distance *d*) with each sample point giving rise to a different value of the objective function.

The number of initial designs and the number of parameters varied for each design can be large and thus gives rise to many solutions and large amounts of data. All of these solutions – if good or poor – may yield valuable information for future simulations and need to be preserved.

If we store this information only in a file system as is common practice at present, it can be very difficult to find data relating to specific simulation runs. Particularly in the post-processing stage of the obtained data, there often is no possibility to search for an individual data range, user description. Storing data in databases provides much more flexibility and builds on the rich and powerful capabilities they provide. Furthermore, when running simulations on the Grid, the parameters for each simulation can be requested from a database system. This effectively allows for computational steering while the simulation is running by changing the parameters stored in the database.

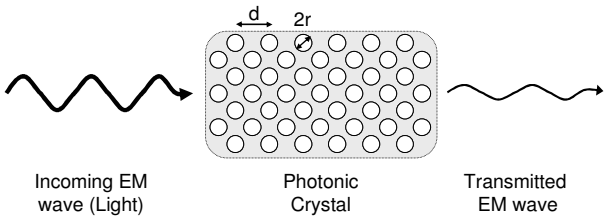

Figure 4 Electromagnetic wave transmission in a photonic crystal. The amplitude of the transmitted light changes according to the design of the crystal. The design parameters in this example are the distance of the holes, *d*, and their radius, *r*.

In this example we shall add a number of PBG designs and populate the database. An Engineer creates a new Grid proxy certificate with his/her credentials, then defines the problem and populates the database with a number of PBG designs. The computation might be done locally or on the Grid by using the Geodise computational toolkit [6]. The following is a script to create data and archive it; '%' represents a comment.

```
% create proxy for access to Grid resources
gd_createproxy;

% define design metadata common to all designs
m.model = 'pbg_design';

% do computation for 10,000 design points
for i=1:10000

  % define random design point in parameter space
  [m.param.radius, m.param.distance] = designpoint(i);
  infile  = ['geometry_', num2str(N), '.cad'];
  outfile = ['spectrum_', num2str(N), '.dat'];
```

www.manaraa.com

```
  % call routine that creates geometry file                    (A)
  create_pbg_geometry_file( infile, m.param );
  % archive geometry definition and metadata                   (B)
  gd_archive( infile, m );

  % Grid compute frequency spectrum for design
  compute_pbg( m.param, infile, outfile );

  % post-process result and obtain bandgap
  m.result.bandgap = postprocess_pbg( outfile );

  % archive spectrum results and metadata                       (B)
  gd_archive( outfile, m );
end
```

Query results: After the computations have finished and the design results are available in the database, the Engineer checks the results with a simple query:

```
% find all PBG designs with bandgap bigger than 99.7
M = gd_query( 'model = pbg_design & result.bandgap > 99.7' )    (C)
M: 4x1 struct array with fields
    standard, access, model, param, result
```

Now, `M` is a vector of structures containing the metadata of all PBG designs with bandwidth larger than 99.7. In this case, four designs match the query.

Retrieve results: For further investigation or visualization a plot for visual comparison, the Engineer decides to retrieve files associated with the above four designs to the local file system.

```
gd_retrieve( {M.standard.fileID}, '/home/Eng007/pbg_files/' )   (D)
display_freqSpectrum('/home/Eng007/pbg_files/*' );
```

The letters (A)-(D) as indicated in the code correspond to the implementation details from Figure 2. Figure 5 shows typical data we can obtain for the various design parameters. The simulation results form an objective function landscape of the photonic bandgap and are the basis from which a full design search may be performed. The storage of the results in a database as well as the transfer of files to a file store on the Grid allow for a design search, design optimisation and re-use by engineers at a later stage through the same transparent Matlab interfaces without requiring knowledge of database technologies or specific query languages.

5. Conclusions and Future Work

We have described a framework which allows the use of distributed databases on the Grid in an engineer-friendly environment. We have implemented a suite of services using an architecture which combines a commercial PSE (Matlab) with a core framework of open standards and service oriented technologies, namely Grid computing, Web services, XML and databases. With a specific example we have shown how design search in electromagnetics can be supported by the Geodise database toolkit. The transparent integration of database tools into the engineering software environment constitutes a starting point for database applications in

www.manaraa.com

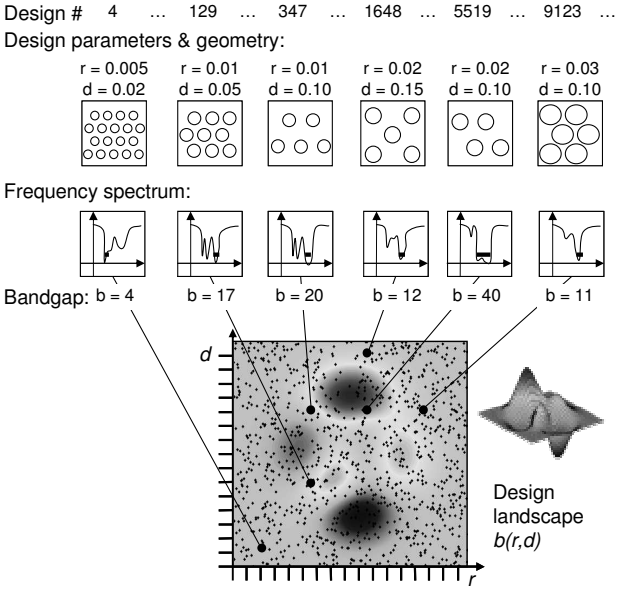

Figure 5 Shown are the relations between the initial design geometries (derived based on a variation of parameters r and d), the computed frequency spectrum, the bandgap obtained by post-processing the frequency spectrum, and the final objective function landscape (figure shows representative data; dots indicate sample points in the parameter space).

engineering design optimisation, and is only one of many potential applications in the engineering domain (CFD, CEM, Civil Engineering, etc.).

The functions we have implemented to extend the Matlab environment allow engineers to share and re-use data conveniently from existing scripts. The automatic generation of standard metadata and support for user-defined metadata allows queries to be formed that represent the Engineer's view of the data. Data of interest can be searched easily using either the Web query interface or the `gd_query` command.

Engineering data usually has a nature of complex and nested structure which can be better represented in XML rather than in the relational data model. Part of our future work will involve providing tools to define XML Schemas that describe custom metadata defined by users. XML Schemas allow us to verify the data to reduce errors, categorise XML documents to identify similar data, create graphical query interfaces for custom metadata, and design an efficient storage strategy, possibly integrated with ontologies developed by Geodise [3] for EDSO. Geodise will provide a graphical user interface to help engineers constructing their workflows, which will need to interact with databases to provide users with up-to-date information. We plan to extend our existing database toolkit to support the workflow construction interface. We will also evolve our Web service based components to OGSA-DAI compliant Grid services, which are a combination of Web services and Grid technology described in the Open Grid Services Architecture (OGSA) [1], the next generation of the Globus Toolkit.

www.manaraa.com

Acknowledgements

This work is supported by the Geodise e-Science pilot project (UK EPSRC GR/R67705/01) and the GEM DTI-funded project. The authors are grateful for many helpful discussions with the Geodise team, and researchers from the myGrid e-Science project team (UK EPSRC GR/R67743/01).

References

[1] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
[2] Geodise Project - Grid Enabled Optimization and Design Search for Engineering, http://www.geodise.org/
[3] L. Chen, N. R. Shadbolt, F. Tao, S. J. Cox, A. J. Keane, C. Goble, A. Roberts, P. Smart. Engineering Knowledge for Engineering Grid Applications, Euroweb 2002, p12-24, 2002.
[4] M.P. Atkinson, V. Dialani, L. Guy, I. Narang, N.W. Paton, D. Pearson, T. Storey and P. Watson. Grid Database Access and Integration: Requirements and Functionalities. UK e-Science Programme Technical Report http://www.cs.man.ac.uk/grid-db/papers/DAIS:RF.pdf
[5] The Storage Resource Broker, http://www.npaci.edu/DICE/SRB/
[6] Web Services Activity, http://www.w3.org/2002/ws/
[7] Matlab 6.5. http://www.mathworks.com
[8] G. Pound, H. Eres, J. Wason, Z. Jiao, A. J. Keane, and S. J. Cox, A Grid-enabled Problem Solving Environment (PSE) For Design Optimisation Within Matlab. To appear in - IPDPS-2003, April 22-26, 2003, Nice, France.
[9] The Globus Project. http://www.globus.org/
[10] M. Molinari. XML Toolbox for Matlab, GEM/Geodise Technical Report, 2002.
[11] J. Hugunin. Python and Java: The Best of Both Worlds. 6th International Python Conference. San Jose, California, USA, 1997.
[12] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H.F. Nielsen. SOAP Version 1.2, W3C Candidate Recommendation, 2002
[13] Commodity Grid Kits. http://www.globus.org/cog/
[14] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. National-Scale Authentication Infrastructure, *IEEE Computer*, 33(12):60-66, 2000.
[15] GridFTP, http://www.globus.org/datagrid/gridftp.html
[16] M. Mealling, P. J. Leach and R. Salz. A UUID URN Namespace, IETF, October 2002. http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-00.txt
[17] IETF PKIX Working Group. http://www.imc.org/ietf-pkix/
[18] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0, November 1996.
[19] Xindice, http://xml.apache.org/xindice/
[20] Oracle9i Database, http://otn.oracle.com/products/oracle9i/content.html
[21] IBM DB2 Version 8.1, http://www-3.ibm.com/software/data/db2/udb/
[22] Microsoft SQL Server 2000, http://www.microsoft.com/sql/
[23] B. Collins, A. Borley, N. Hardman, A. Knox, S. Laws, J. Magowan, M. Oevers, E. Zaluska. Grid Data Services - Relational Database Management Systems (Version 1.1), DAIS-WG Document, 2002.
[24] A. Krause, K. Smyllie, and R. Baxter. Grid Data Service Specification for XML Databases, DAIS-WG Document, 2002.
[25] Global Grid Forum, http://www.ggf.org/
[26] Matlab Database Toolbox, http://www.mathworks.com/products/database/
[27] The GEM project, http://www.soton.ac.uk/~gridem

www.manaraa.com