A Grid-Enabled Problem Solving Environment (PSE) for Design Optimisation within Matlab

G. E. Pound, M. H. Eres, J. L. Wason, Z. Jiao, A. J. Keane, and S. J. Cox
School of Engineering Sciences, University of Southampton, UK
{gep, eres, j.l.wason, z.jiao, ajk, sjc}@soton.ac.uk

Abstract

The process of design search and optimisation using Computational Fluid Dynamics (CFD) is computationally and data intensive, a problem well-suited to Grid computing. The Geodise toolkit is a suite of Grid-enabled design optimisation and search tools within the Matlab environment. The use of these tools by the engineer is facilitated by intelligent design advisers targeted initially at CFD. The role of remote computation and data access in constructing a Grid-enabled Problem Solving Environment is discussed. The use of the Geodise toolkit for design optimisation from within the Matlab environment is considered with an exemplar problem.

1. Introduction

The process of design search and optimisation involves the modelling and analysis of engineering problems to yield improved designs. Design parameters that the engineer wishes to optimise are identified, and a measure of the quality of a particular design (the objective function) is computed using an appropriate model. A number of design search algorithms may then be used to yield more information about the behaviour of a model over the parameter space, and to minimise/maximise the objective function to improve the quality of the design. This process may include lengthy and repetitive calculations to obtain the value of the objective function with respect to the design variables.

Design optimisation with regard to fluid dynamics is relevant to, amongst others, the aerospace, automotive and oil industries. Computational Fluid Dynamics (CFD) allows the engineer to analyse the properties of a design. However, detailed analysis is computationally expensive. To perform the numerous solutions required for extensive parameter exploration during a design search in this domain normally requires access to significant computational resources.

The computationally intensive problem domain of engineering design optimisation using CFD can be well matched to the field of Grid computing using appropriate search methods. Grid computing addresses the technologies and infrastructure to allow large-scale resource sharing and facilitate the performance of virtual organisations (VOs) that form to solve science and engineering problems [1]. Fundamental to Grid computing is the exposure and discovery of remote resources, in particular compute and data resources, that may represent a heterogeneous mix of technologies.

The Geodise project [2] aims to aid the engineer in the design process by making available a suite of design optimisation and search tools and CFD analysis packages integrated with distributed Grid-enabled computing, data, and knowledge resources. Facilitating the use of such design search tools requires the integration of intelligent design advisers, which are able to support the engineer throughout the design process by providing ontology services, annotation services [3], and context sensitive advice based on the states of the computation.

Engineering design search is also data intensive, and data may be generated at different locations with different characteristics. It is often necessary for an engineer to access a collection of data produced by design and optimisation processes to make design decisions, perform further analysis and carry out post-processing. Databases are valuable to expose the state of the design process to context sensitive design advisers, allowing them to provide dynamic advice to the user. Databases therefore play an essential role in our architecture, where it is important to capture the process of how results are obtained in addition to storing the results themselves.

Traditionally, data in many scientific and engineering disciplines have been organised 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. Here we focus upon providing data management in an engineering environment by leveraging existing database tools that are not commonly used in this field, and making them accessible to users of the system. We use databases to store, maintain and access data associated with result and intermediate files, and authorisation control to improve the accessibility of the data and to encourage collaborations among the engineers. An important issue regarding data sharing in a VO is data access control, which includes authentication and authorisation. Authentication involves verifying the claimed identity of a user, whereas authorisation checks an authenticated

users access rights for specific data and computational resources. A consistent access control mechanism for all the resources in a VO is required.

We adopt a service approach for database integration into a Grid environment, providing other Grid applications with a well-defined interface for accessing and archiving data. The Data Access and Integration Services (DAIS) Working Group of the Global Grid Forum (GGF) [5] are developing requirements, functionalities and standards for Grid Database Services [4][6] in the Open Grid Services Architecture (OGSA) [7]. The DAIS activities were initiated by the UK e-Science Programme Database Task Force [8]. In the future each of our Geodise specific data management services will communicate with the underlying databases through such services.

All these objectives impose a number of requirements upon our choice of environment. The environment should provide an intuitive interface to the available Grid resources. A Grid-enabled Problem Solving Environment (PSE) abstracts the complexities of accessing the Grid by providing a complete suite of high level tools designed to tackle a particular problem area [9]. Nimrod/G [10] is a tool that facilitates parameter studies over computational Grids. Triana [11] is a graphical programming environment which abstracts the complexities of composing distributed workflows.

Whilst it is possible to reduce the complexity of the technologies faced by the user, it is important that the environment chosen has the flexibility to tackle the subtleties of a wide range of workflows within the problem domain. A previous prototype that consisted of a wizard style web portal that guided the user through a design optimisation problem [12] proved inflexible, and the wizard would not scale to encompass a much larger number of problems. The complexity and variation of the workflows involved in the design process, mean that a scriptable environment that allows the user to tailor workflows to the task in hand is valuable.

Here the user interface used to expose the functionality provided by the Geodise PSE is the commercial Matlab environment [13]. The Matlab package provides a fourth generation language for numerical computation, built-in math and graphics functions and numerous specialised toolboxes for advanced mathematics, signal processing and control design. The Matlab product is widely used in academia and industry to prototype algorithms, and to visualise and analyse data. Matlab 6.5 also features a number of 'just-in-time' acceleration technologies to increase the performance of native Matlab code.

The rationale behind adopting Matlab as the user interface for the Geodise PSE is pragmatic. As a toolkit that may be integrated into an environment routinely used by our industrial and academic partners the Geodise PSE becomes a flexible tool, part of the engineer's arsenal. The NetSolve system [14] which uses a client-server architecture to expose platform dependent software libraries has also successfully adopted Matlab as a user interface.

The final Geodise toolkit will be composed of a hierarchy of components. Low level compute and database functions will be available to the user, in addition to powering a number of higher level design search, pre/post-processing, and CFD functions. All of these components will be available through a suite of intelligent design advisers that will guide the user through the design process, and facilitate the use of toolkit components.

The remainder of this paper focuses on the process of exposing Grid enabled resources to the Matlab environment, allowing us to compose the required low level compute and data access functionality. We first describe the architecture and the functionality of the computation and database components of the toolkit. We then demonstrate the use of these functions in an example iteration of the design process.

2. Geodise Toolkit

The fundamental technologies behind the compute and data components of the Geodise toolkit are currently distinct and are described below.

2.1. Computation

The user of the Geodise toolkit acts as a client to remote compute resources that are exposed as Grid services. Users should be authenticated, and then authorised to access resources to which they have rights. They need to be able to submit their own code to compute resources, or run software packages that are available as services. The user should be able to discover the available resources, decide where to run a job and be able to monitor its status. It is essential that the user be able to easily retrieve the results of a simulation. Additionally the requirements of design search and optimisation mean that compute resources must be available programmatically to algorithms that may initiate a large number of computationally intensive jobs serially or in parallel.

The Globus toolkit [15] provides middleware that allow the composition of computational grids through the agglomeration of resources which are exposed as Grid services. This middleware provides much of the functionality required by our toolkit including authentication and authorisation (GSI), job submission (GRAM), data transfer (GridFTP) and resource monitoring and discovery (MDS).

Client software to Globus Grid services exists natively on a number of platforms and also via a number of

Commodity Grid (CoG) kits [16] that expose Grid services to 'commodity technologies'; including Java [17], Python, CORBA [18], and Perl. By using client software to Grid services written for these commodity technologies the developer of a PSE is able to remain independent of platform and operating system.

The independence allowed by adopting a commodity technology motivated development of the Geodise toolkit over the Grid service client APIs of the Java CoG kit v.0.9.13 [17]. Java [19] is a mature technology that runs compiled byte-code within a virtual machine. The Matlab environment itself runs within a Java Virtual Machine (JVM), and provides an external interface which allows Java classes to be instantiated and invoked easily within the Matlab workspace. The support of Java version 1.3.1 by Matlab 6.5 provides the utility which makes the Java language suitable for programming Grid middleware.

The Java CoG provides a number of low-level mappings, in the form of a number of Java packages, which are APIs to the respective Globus Grid service clients. To expose the functionality available from the Java CoG to the Matlab user it was important to present functions which are consistent with the behaviour and syntax of the Matlab environment. Functions are written in the interpretive Matlab language, and these call Java classes which access the Java CoG API. Functions are written with the intention that they may be incorporated programmatically into the higher level components of the toolkit.

Table 1 describes the compute functions used by the Geodise 0.3 system. This set of functions describes the minimum functionality required to allow the user to run jobs on Globus compute resources. The functions may be loosely categorised into those concerned with the user's credentials, job submission to the Globus Resource Allocation Manager (GRAM), and file transfer.

Table 1: Compute commands.

Function Name	Description
`gd_createproxy`	Creates a Globus proxy certificate from the user's credentials
`gd_jobsubmit`	Submits a GRAM job, specified by a RSL string, to a Globus server. Returns a job handle to the user.
`gd_jobstatus`	Returns the status of the GRAM job specified by a job handle.
`gd_jobkill`	Terminates the GRAM job specified by a job handle.
`gd_listjobs`	Returns job handles for all GRAM jobs associated with the users credentials registered on a MDS server.
`gd_getfile`	Retrieves a file from a remote host using GridFTP.
`gd_putfile`	Transfers a file to a remote host using GridFTP.

The Grid Security Infrastructure (GSI) [20] used by the Globus toolkit is based upon the Public Key Infrastructure (PKI) [21]. Under the PKI an individual's identity is asserted by a certificate that is digitally signed by a Certificate Authority within a hierarchy of trust. In an extension to this standard the GSI allows a user to delegate their identity to remote processes using a temporally limited proxy certificate signed by the user's certificate. The toolkit command `gd_createproxy` allows a user to create a Globus proxy certificate within the Matlab environment, essentially creating a point of single sign-on to the Grid resources that the user is entitled to use.

The `gd_jobsubmit` command allows users to submit compute jobs to a GRAM job manager described by a Resource Specification Language (RSL) string. The `gd_jobsubmit` command returns a unique job handle which identifies the job. The job handle may be used to query or terminate the user's job. In addition the `gd_listjobs` command may be used to query a Monitoring and Discovery Service (MDS) to return all the job handles associated with the user's certificate.

Two file transfer commands are provided to allow users to transfer files to and from Grid-enabled compute resources to which they have access. These commands support the high performance file transfer protocol GridFTP [22]. The GridFTP protocol defines a number of extensions to the FTP protocol to enable transfer of high volumes of data.

2.2. Database

Users need a simple, transparent way to store files in a repository along with additional information (metadata) about those files which will make it easier for members of a VO to find the files at a later date and use them effectively. This metadata should be generated automatically where possible and be provided by the user when necessary. They should be able to specify who else can discover and retrieve these files. A query mechanism should be available with facilities for interactive and non-interactive use, so that files can be located programmatically in scripts. The engineer should be able to specify what kind of file they are looking for in an intuitive manner without necessarily needing knowledge of a database query language (e.g., SQL). Ideally the user should not need to know the name of the database or machine their data is stored on, or its underlying storage mechanism. It should also be possible to locally record a unique identification number for accessing the file once it has been archived and use that handle at a later date to retrieve the file.

A simulation or optimisation may take a long time and rather than ever repeat parts of the process it would be advantageous to store files for future reuse. For

performance reasons it is desirable to store files close to where they will be used the most, usually the site where they were produced. However, accessibility by users at other sites in a VO should also be considered if collaboration and sharing is to be encouraged. A secure and reliable transport mechanism is required and GridFTP meets both of these requirements. Unique IDs are used to prevent files belonging to different users being overwritten. A file location service keeps a record of file IDs and locations in a database so that the unique identifier is all that is required in order to locate the file.

When there are numerous result files in various locations it becomes difficult to know which ones to retrieve when needed and even harder to share them. A solution to this problem is to store and retrieve data files based on additional descriptive metadata, for example standard file metadata such as file name and size, and application specific metadata such as the number of variables used in an optimisation, and their description or numerical values. Providing a way to find files of interest based on their characteristics rather than their location in a file system gives users a more effective way to manage their own data, provides a means for reuse and collaboration, and may act as a source for advice based on similar examples of a given problem.

In our architecture we use relational databases for structured data such as authorisation information because they are mature, reliable and scalable. Relational databases also have a well defined standard interface which allows the development of generic tools for a number of operations such as creating interfaces and constructing queries. We have also chosen XML [23] repositories for more flexible storage of complex, deeply nested engineering application specific metadata. We therefore 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 [24] and XML repositories [25] and we will follow these projects closely and use implementations that follow the proposed standards from OGSA – DAIS [6].

In Geodise 0.3 we have implemented tailored web services that provide API interfaces to specific databases. A web service is a self-describing programmable component that can be discovered and invoked over the web. The web service interface is described in a standard format using the Web Services Description Language (WSDL) [26]. Methods specified in the WSDL may be invoked using the Simple Object Access Protocol (SOAP) [27], which uses a combination of XML and HTTP to transfer data between web services regardless of their underlying programming language or platform. One example from the Geodise 0.3 implementation is the ability of a Java client code running on Linux to communicate, using the Apache SOAP API [28], with .NET web services on a Microsoft server.

The metadata service provides a means for metadata to be stored in a database, queried and retrieved by client programs. The metadata service must manage a combination of standard and application specific custom metadata. The existing system uses relational databases to store standard metadata and an XML repository for custom metadata, as it is a more extensible option. The user specifies their custom metadata as a Matlab structure which is then converted into XML using the XML toolbox for Matlab [29] and sent to the metadata service for storage. Similarly, when a query is performed the XML metadata results are converted back into a structure before displaying them to the user.

An interactive, graphical query interface is also provided in which a user specifies their selection criteria in a web form generated based on the metadata structure. In this interface there is an option to generate Matlab code needed for retrieving selected files that match the criteria, which can then be pasted into the user's own Matlab script.

In our current implementation, the authentication of a user is achieved by using GSI [20]. Authorisation is implemented as a service which uses a database of registered users to keep track of permissions on data and map between Globus certificate subjects and user IDs. Authorisation exists at different levels of granularity and must be consistent for metadata and files. A user is first assigned a role which determines their access to the repository, for example certain roles can write data while others may only read data. Role based access models are important for collaborative working, when the individual performing a role many change over time and when several individuals may perform the same role at the same time [4][30]. A user's read access to the repository is restricted to their data and that of any other user who has granted them permission.

Table 2 describes the Geodise 0.3 database functions. These functions provide users with the ability to store files in a repository with associated metadata, query the metadata and retrieve the files, providing they have the correct access rights.

Table 2: Database commands.

Function Name	Description
`gd_archive`	Stores a file in a repository with associated metadata.
`gd_retrieve`	Retrieves a file from the repository to the local machine.
`gd_query`	Retrieves metadata about a file based on certain restrictions.

The gd_archive function will store a given file in a repository for an authenticated user. The function is able to generate a structure containing some standard metadata for the file, such as its local name, size, format, and creation time. The user may add additional metadata, for example comments, custom information specific to that format, and a list of users or groups who may access the file. The function then transports the file to a server using GridFTP and also sends the metadata to a database accessed via a web service. The gd_archive function returns a unique handle which can be used to retrieve the file at a later date.

The metadata that is stored can be queried by an authorised user with the gd_query command, in order to discover files that have certain characteristics and obtain information about them, such as their handle for retrieval. The gd_retrieve function will locate a file based on a given file handle and return it to a local directory.

3. Geodise 0.3 Application Exemplar

To demonstrate the possible use of our Grid-enabled Matlab toolkit we choose a basic problem of fluid dynamics, which is the two dimensional, external, laminar flow over a NACA four digit airfoil. A sketch of the problem is given in Figure 1. At the velocity inlet the assumed free-stream velocity profile is constant, and the angle of attack is measured in the counter-clockwise direction to the horizontal. The upper and lower boundaries are periodic, and we have a pressure outlet on the right hand side of the computational domain. The airfoil profiles are generated by using standard NACA four digit expressions [31].

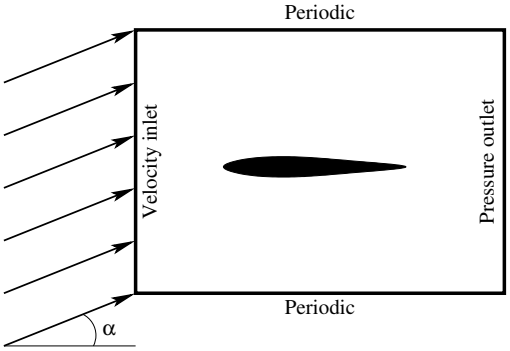

Figure 1. NACA four digit airfoil problem with boundary conditions. Here α is the angle of attack.

The above mentioned problem can be solved by using various finite element analysis tools. We will use two commercially available codes, i.e. Gambit and Fluent [32], for the mesh generation and solution processes, respectively.

In general the process of obtaining a numerical solution on a remote Globus server involves at least the following steps:

1. The user generates a Grid proxy by entering their password.
2. The user generates data and input files required for the mesh generation and analysis software, and transfers them to the remote Globus server.
3. The mesh generation and analysis tools are run on the Globus server and intermediate files are queried to retrieve information regarding the mesh quality and the convergence of the solution.
4. If everything seems satisfactory the user transfers the simulation results to the local file system, checks the objective function values and possibly visualizes the simulation results in Matlab, or a third party product/plug-in.

Since the Grid proxy is required to use the Grid-enabled resources, the user initiates their Grid proxy certificate by using the gd_createproxy command. This command invokes the Java CoG, which in turn pops a window where the user can enter their password. After the user enters their password and presses the "Create" button, a proxy certificate is generated for the user.

Now, the user generates the vertex file of a NACA four digit airfoil, which is simply a text file containing the coordinates of the airfoil. The vertex file is transferred to the remote Globus server where it will be used by the mesh generation tool Gambit.

The next step involves the preparation of journal files for Gambit and Fluent and transferring them to the Globus server. These journal files are tailored according to various input parameters which are entered by the user. A Gambit journal file informs Gambit to use the vertex file as the input file, to mesh the domain by using a given mesh size parameter, and to export a Fluent compatible mesh file as output. Similarly, the Fluent journal file instructs that program to use the mesh file as input, to use inlet velocity and angle of attack parameters in the numerical solution, and to export a data file after the solution converges. When the journal files are ready, the user transfers them to the remote Globus server by using the gd_putfile(<FQHN>, <Local File>, <Remote File>) Matlab command. Here, <FQHN> is the fully qualified host name of the remote Globus server. A snapshot of this step is shown in Figure 2. These steps can be categorized as the initial data preparation and transfer, and a flowchart of this process is given in Figure 3.

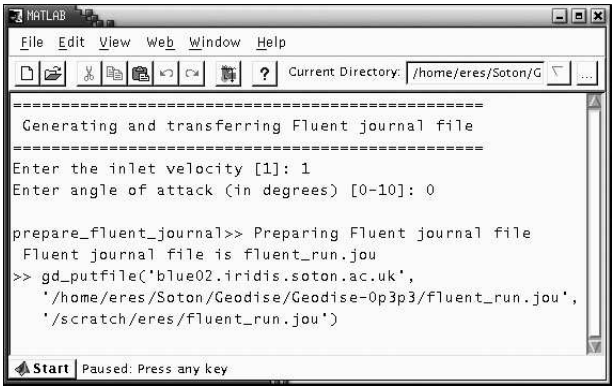

Figure 2. Generating a Fluent journal file in Matlab environment, and transferring it to a Globus server. Here, the user inputs are the inlet velocity and the angle of attack.

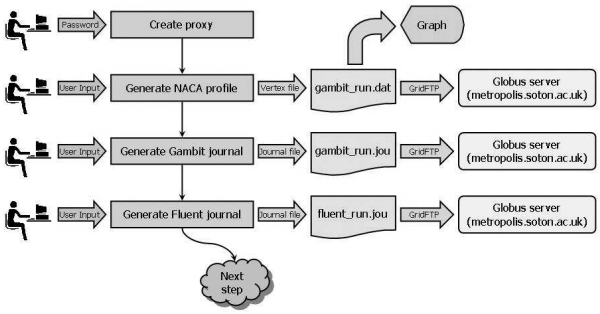

Figure 3. Initial data flow during the generation of the Grid proxy, and preparation of data and journal files.

After the initial data preparation step is complete the user can submit their jobs to the remote Globus server by using the `gd_jobsubmit(<RSL>, <FQHN>)` command. Here, the `<RSL>` describes the name of the executable on the Globus server, the executable's command line arguments, the names of standard input, output and error files, etc. After the job submission the `gd_jobsubmit` command returns a job handle back to the Matlab environment, which is later used to check the status of the submitted job.

Mesh generation and analysis steps are also summarized in Figure 4 as a process flowchart. A properly generated mesh file is required by the analysis tool, the user must generate the mesh file by submitting the geometry to the Gambit mesh generation tool. The user must wait until Gambit finishes meshing, and the Globus server changes its status from 'ACTIVE' to 'DONE'. Additionally, the user needs to make sure that the mesh generation process succeeds, and the quality of the generated mesh is acceptable for the analysis tool. Therefore, before running the analysis tool the state of the mesh generation, and mesh quality are checked by transferring the standard error file of Gambit to the local file system and parsing the mesh quality information from that file.

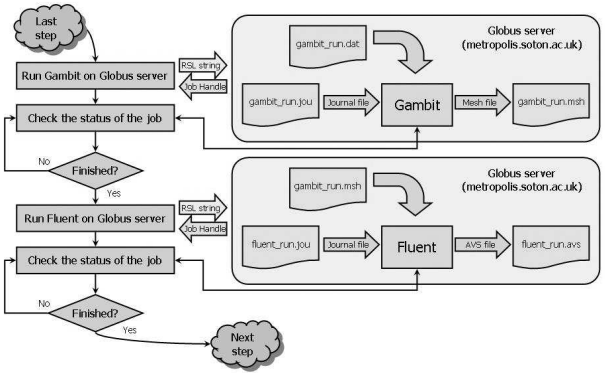

Figure 4. Process flow diagram for mesh generation and analysis tools.

If everything is satisfactory, the user now can submit the analysis job on the remote Globus server, get back a job handle, check the status of the job, and retrieve convergence information and objective function values by using a very similar process (Figure 5 and Figure 6). Finally, the solution can be transferred back to the local file system and visualized in Matlab (Figure 7). Throughout this process intermediate and solution files can be archived in the Geodise repository, with the `gd_archive` command. By associating metadata with the files the design archive may be accessed interactively or programmatically when required using `gd_query` (Figure 8).

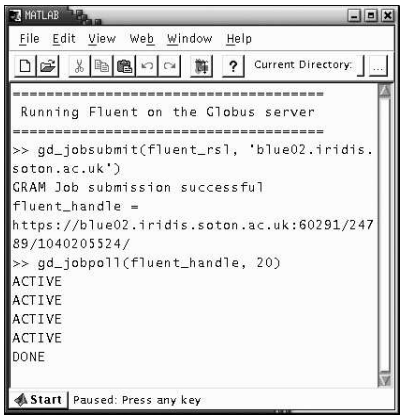

Figure 5. Running Fluent on the Globus server by using a proper RSL string and previously generated journal file. The job status is polled every ten seconds.

If design search is being carried out this process will be repeated as new and possible improved designs are considered. If Design of Experiment methods are being used to create response surface models [33] multiple runs may be scheduled in parallel and the resulting data archive used to study the design problem.

Figure 6. Transferring Fluent output file to the local file system, and parsing it to retrieve objective function values.

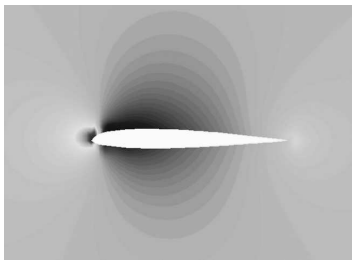

Figure 7. Visualizing the data file in Matlab environment.

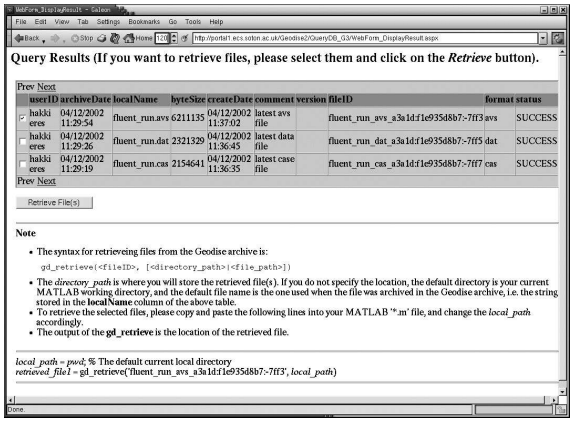

Figure 8. The Web interface showing query results from the database.

4. Conclusions and Future Work

The choice of the Matlab environment appears to be a pragmatic decision, providing a flexible and robust user interface for Grid computing. The Java CoG [17] provides a valuable platform independent client side API for the access of Globus Grid-enabled resources. Database technologies have an important role to play in engineering design and optimisation and are suitable for managing contextual and technical metadata associated with the design processes, and facilitate data sharing among engineers. The use of web services for API access to databases is beneficial for creating usable client side functions and facilitating communication between different languages and platforms.

By exposing the compute and data functionality as toolkit components we are able to construct high level functions which utilise Grid resources for CFD and design search tasks. Given commands in a high level interpretive language it is straightforward for the engineer to exploit available Grid-enabled resources to tackle computationally and data intensive tasks.

Future work on this project will focus on the creation of high level application components. This work will include the exposure of heterogeneous legacy code to the PSE. Refining the existing low-level compute components will involve exposing additional client side tools to allow the user to discover the available compute resources, and to make an informed decision where to submit compute jobs. Support for high-performance third party file transfer will be included.

A future requirement for a fault tolerant data management system is the provision of a local personal metadata archive which replicates the data stored in the main repository so that users are still able to locate and retrieve their own files if the central metadata service or external network is down. In order to preserve consistency between files and their metadata, updates must be controlled through the Geodise API. Users should be prevented from removing or overwriting files in the repository by any other means, for example using GridFTP directly. Data lifetime management is another issue, i.e., a mechanism is needed to specify how long a collection of data will be stored in Geodise repository, and be able to extend the lifetime, or perform clean up tasks.

We expect that the architectures of the computational and database components will converge with a move to an Open Grid Services Architecture (OGSA) [7] model. The implementation of OGSA defines a number of extensions to standard XML web services that provide the common functionality required by both the computational and database components.

Acknowledgements

This work is supported by the GEODISE e-Science pilot project (UK EPSRC GR/R67705/01). The authors gratefully acknowledge many helpful discussions with the GEODISE team, and researchers from the myGrid e-Science project team (UK EPSRC GR/R67743/01). We thank Fluent, Microsoft and Intel for ongoing support.

References

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organisations, *International Journal of Supercomputer Applications*, 15(3):200-222, 2001.

[2] The Geodise Project. 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, *Proceedings of the Euroweb 2002: The Web and the GRID: from e-science to e-business*, 12-24, 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, *Database Access and Integration Services Working Group Document*, 2002.

[5] Global Grid Forum. http://www.gridforum.org/

[6] A. Krause, S. Malaika, G. McCance, J. Magowan, N. W. Paton, and G. Riccardi. Grid Database Service Specification, *Database Access and Integration Services Working Group Document*, 2002.

[7] 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 Working Group Document*, 2002.

[8] N. Paton, M. Atkinson, V. Dialani, D. Pearson, T. Storey, and P. Watson. Database Access and Integration Services on the Grid, *UK e-Science Programme Technical Report Series*, 2002.

[9] G. von Laszewski, I. Foster, J. Gawor, P. Lane, N. Rehn, and M. Russell. Designing Grid-based Problem Solving Environments and Portals, *34th Hawaiian International Conference on System Science*, 2001.

[10] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modelling with Nimrod/G: A Killer Application for the Global Grid, *Proceedings of the International Parallel and Distributed Processing Symposium*, 520-528, 2000.

[11] Triana. http://www.triana.co.uk/

[12] S. J. Cox. Grid Enabled Optimisation and Design Search for Engineering (GEODISE), *NeSC Workshop on Applications and Testbeds on the Grid*, 2002.

[13] Matlab 6.5. http://www.mathworks.com/

[14] H. Casanova and J. J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems, *International Journal of High Performance Computing Applications*, 11(3):212-223, 1997.

[15] The Globus Project. http://www.globus.org/

[16] Commodity Grid Kits. http://www.globus.org/cog/

[17] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit, *Concurrency and Computation: Practice and Experience*, 13(8-9):643-662, 2001.

[18] M. Parashar, G. von Laszewski, S. Verma, J. Gawor, K. Keahey, and N. Rehn. A CORBA Commodity Grid Kit, *Concurrency and Computation: Practice and Experience* (to appear), 2002.

[19] Java 2. Sun Microsystems Inc., http://java.sun.com/

[20] 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.

[21] IETF PIKX Working Group. http://www.imc.org/ietf-pkix/

[22] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing, *IEEE Mass Storage Conference*, 2001.

[23] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0, *W3C Recommendation*, 1998.

[24] 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), *Database Access and Integration Services Working Group Document*, 2002.

[25] A. Krause, K. Smyllie, and R. Baxter. Grid Data Service Specification for XML Databases, *Database Access and Integration Services Working Group Document*, 2002.

[26] R. Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana. Web Services Description Language (WSDL) 1.2, *W3C Working Draft*, 2002.

[27] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H.F. Nielsen. SOAP Version 1.2, *W3C Working Draft*, 2002.

[28] Apache SOAP. http://xml.apache.org/soap/

[29] M. Molinari. XML Toolbox for Matlab (Version 1.0), *GEM/Geodise Technical Report*, 2002.

[30] D. Pearson. Data Requirements for The Grid: Scoping Study Report, *Database Access and Integration Services Working Group Document*, 2002.

[31] I. Abbott and A. von Doenhoff. *Theory of Wing Sections.* Dover Publications, Inc., New York, 1959.

[32] Fluent Web site. http://www.fluent.com/

[33] D. Jones, M. Schonlay, and M.Welch. Efficient Global Optimization of Expensive Black Box Functions, *Journal of Global Optimisation*, 13:455-492, 1998.