Implementation of a Grid-enabled Problem Solving Environment in Matlab

Hakki Eres, Graeme Pound, Zhouan Jiao, Jasmin Wason, Fenglian Xu, Andy Keane, and Simon Cox

School of Engineering Sciences,
University of Southampton,
Highfield, Southampton, SO17 1BJ, UK
{eres, gep, z.jiao, j.l.wason, f.xu, ajk, sjc}@soton.ac.uk
http://www.geodise.org/

Abstract. In many areas of design search and optimisation one needs to utilize Computational Fluid Dynamics (CFD) methods in order to obtain numerical solution of the flow field in and/or around a proposed design. From this solution measures of quality for the design may be calculated, which are required by optimisation methods. In large models the processing time for the CFD computations can very well be many orders of magnitude larger than the optimisation methods; and the overall optimisation process usually demands a combination of computational and database resources; therefore this class of problems is well suited to Grid computing. The Geodise toolkit is a suite of tools for Grid-enabled parametric geometry generation, meshing, CFD analysis, design optimisation and search, database, and notification tools within the Matlab environment. These grid services are presented to the design engineer as Matlab functions that conform to the usual syntax of Matlab. The use of the Geodise toolkit in Matlab introduces a flexible and Grid-enabled problem solving environment (PSE) for design search and optimisation. This PSE is illustrated here 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 [1]. Independent design parameters that the engineer wishes to modify, and relevant design constraints 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 by modifying the design parameters subject to any constraints imposed on them. 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) methods allow the engineer to analyse the properties of a design. However, they often require computer aided design (CAD) tools to generate parametric design geometries, mesh generation programs to mesh the flow domain, and CFD solvers to obtain an approximate solution for each design search point. Therefore, the modeling and analysis is usually the computationally expensive part of design search and optimisation process. To perform the numerous solutions required for extensive parameter exploration during a design search in this domain normally requires access to significant computational and database resources.

Engineering design search and optimisation is also a data intensive process. Data may be generated at different locations with different characteristics. It is often necessary for a design 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.

The Geodise project [2] aims to aid the engineer in the design process by making available a suite of intelligent design optimisation and search tools, CAD packages, mesh generation tools, and CFD analysis packages integrated with distributed Grid-enabled computing, analysis, 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, and context sensitive advice based on the states of the computation.

A problem solving environment (PSE) provides the user a complete and integrated environment for problem composition, solution, and analysis [3]. PSEs exist for numerous application areas, such as the solution of partial differential equations, scientific visualisation, and computational chemistry. Various parts of the design search and optimisation process can be integrated as a PSE to aid the design engineer during tedious steps. Graphical components with drag-and-drop capabilities can ease the involvement of a novice design engineer; on the other hand, an expert user might require more flexibility which can only be provided through direct access to a scripting language.

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 PSE abstracts the complexities of accessing Grid resources by providing a complete suite of high level tools designed to tackle a particular problem area [4]. There are various ongoing projects for Grid-enabled PSEs, for example, Nimrod/G [5] is a tool that facilitates parameter studies over computational Grids, and Triana [6] 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 [7] that consisted of a wizard style web portal that guided the user through design optimisation problems was useful. However, it proved to be inflexible, because it failed to provide the user with the ability to re-use and re-compose workflows for large-scale problems. The complexity and variation of the workflows involved in typical design processes mean that a scriptable environment where the user can tailor workflows to the task in hand is valuable.

The user interface used to expose the functionality provided by the Geodise PSE is chosen to be the commercial Matlab environment [8]. The Matlab package provides an interpretive language for numerical computation, built-in math and graphics functions and numerous specialised toolkites 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. Additionally, Matlab enables the programmer to access Java classes, to create objects, and to call methods of these objects within the Matlab environment by using Matlab's functions and commands.

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 [9] which uses a client-server architecture to expose platform dependent software libraries has also successfully adopted Matlab as a user interface. It is an examplar of a hosting environment which can be used to couple together a range of grid-services which expose themselves via open-standard protocols.

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 a number of higher level design search and optimisation, 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 geometry and mesh generation, analysis, visualisation, data access, and notification functionality in a scriptable PSE. We first describe the architecture and the functionality of the existing Geodise toolkit. We then demonstrate the use of these functions in an example iteration of the design process.

2 Geodise Toolkit

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 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 [10] 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, job submission, data transfer, and resource monitoring and discovery.

Client software to Globus Grid services exists natively on a number of platforms [11] and also via a number of Commodity Grid (CoG) kits [12] that expose Grid services to 'commodity technologies'; including Java [13], Python, CORBA [14] 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 [13]. Java 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. All functions are implemented in the Matlab language, and they call Java classes which access the Java CoG API. Additionally, these functions are written with the intention that they may be incorporated programmatically into the higher level components of the toolkit.

Table 1 lists the implemented functions in the Geodise toolkit. This set of functions can be loosely categorized as: i) functions which allow the user to run and control jobs on Globus compute resources, ii) functions which are used to archive, query, and retrieve data, and iii) functions which are used to notify the user.

The toolkit command `gd_createproxy` allows a user to create a temporally limited 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. `gd_certinfo`, `gd_proxyinfo`, `gd_proxyquery`, and `gd_destroyproxy` are associated utility commands which are used to query or invalidate the user's proxy certificate. The `gd_jobsubmit` command allows users to submit compute jobs to a GRAM job manager described by a Resource Specification Language

Table 1. Implemented commands in the Geodise toolkit

Function Name	Description
`gd_archive`	Stores a file in a repository with associated metadata
`gd_certinfo`	Returns information about the user's certificate
`gd_createproxy`	Creates a Globus proxy certificate from the user's credentials
`gd_destroyproxy`	Destroys the local copy of the user's Globus proxy certificate
`gd_getfile`	Retrieves a file from a remote host using GridFTP
`gd_jobkill`	Terminates the GRAM job specified by a job handle
`gd_jobpoll`	Queries the status of a Globus GRAM job until complete
`gd_jobstatus`	Returns the status of the GRAM job specified by a job handle
`gd_jobsubmit`	Submits a GRAM job to a Globus server
`gd_listjobs`	Returns job handles for all GRAM jobs
`gd_proxyinfo`	Returns information about the user's proxy certificate
`gd_proxyquery`	Queries whether a valid proxy certificate exists
`gd_putfile`	Transfers a file to a remote host using GridFTP
`gd_query`	Retrieves metadata about a file based on certain restrictions
`gd_retrieve`	Retrieves a file from the repository to the local machine
`gd_sendtext`	Sends a text message to the specified mobile phone number

(RSL) [15] string. The `gd_jobsubmit` command returns a unique job handle which identifies the job. The job handle may be used to query, poll, or terminate the status of the user's job by using `gd_jobstatus`, `gd_jobpoll`, and `gd_jobkill`, respectively. 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, `gd_putfile` and `gd_getfile`, are provided to allow users to transfer files to and from Grid-enabled compute resources. These commands support the high performance file transfer protocol GridFTP [16]. A third-party type of file transfer command will be implemented in the future release of the Geodise toolkit.

Database related functions of the Geodise toolkit 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. 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 in the future. 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` command 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 provides the user with the Matlab code which returns the file to a specified local directory.

For the event notification we use the widely popular Short Message Service (SMS) from mobile telecommunications technology. The Matlab command `gd_sendtext` can send a text message to the specified mobile phone number via a 'pay-per-use' service, independent of its geographical location, and inform the user about the state of the processes. The client and the server of the text messaging service run on Globus servers, hence, they both benefit from the security measures provided by the Globus toolkit. We also use Matlab's native function, `sendmail`, to e-mail the user simulation results by using an SMTP server.

3 Geodise Application Exemplar

To demonstrate the possible use of our Grid-enabled Matlab PSE we choose a basic problem of fluid dynamics, which is the two dimensional, external, laminar flow over a NACA four digit airfoil. A sketch and a sample solution of the problem are given in Fig. 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 there is a pressure outlet on the right hand side of the computational domain. The airfoil profiles are generated by using standard NACA four digit expressions [17]. This problem can be solved using various CFD tools. The Geodise toolkit currently has two commercially available codes: Gambit and Fluent [18], for the mesh generation and solution processes, respectively. Nevertheless it is possible to integrate different codes as long as they can be run on the Globus server.

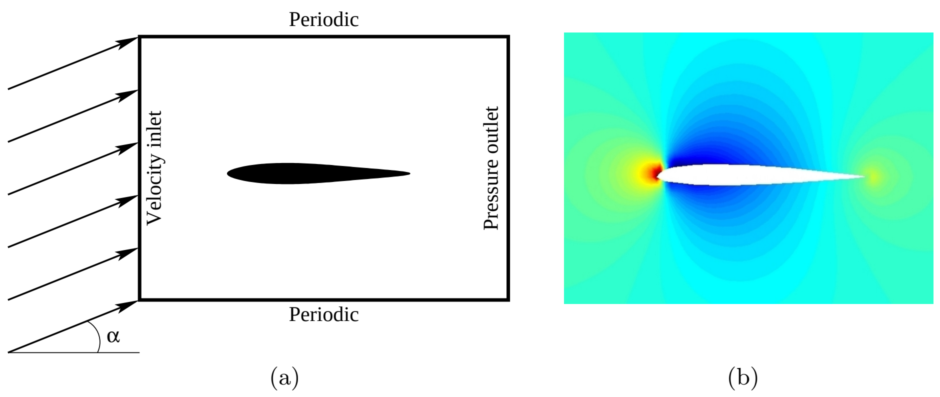
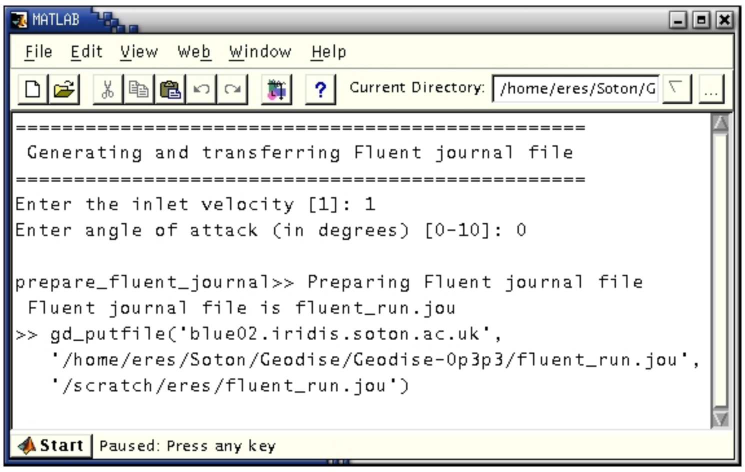


Fig. 1. (a) NACA four digit airfoil problem with boundary conditions. Here, α is the angle of attack. (b) Pressure contours visualized in Matlab

Since a valid Grid proxy is required to use the Grid-enabled resources, the user initiates their Grid proxy certificate by using `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.

The next step involves the preparation of vertex data and journal files for Gambit and Fluent, and their transfer to the Globus server. The vertex data file for a NACA four digit airfoil is a text file containing the coordinates of the airfoil, which are used by the mesh generation tool Gambit. The journal files for Gambit and Fluent are tailored according to various input parameters entered by the user. The Gambit journal file informs Gambit to use the vertex file as the input file, to mesh the domain 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 Fluent journal file preparation and transfer is shown in Fig. 2.

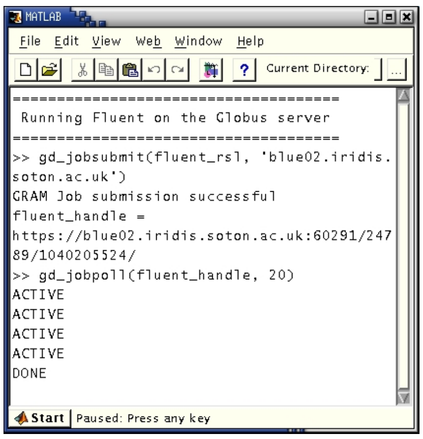

MATLAB
File Edit View Web Window Help
Current Directory: /home/eres/Soton/G
Start
Paused: Press any key

```
================================================
 Generating and transferring Fluent journal file
================================================
Enter the inlet velocity [1]: 1
Enter angle of attack (in degrees) [0-10]: 0

prepare_fluent_journal>> Preparing Fluent journal file
 Fluent journal file is fluent_run.jou
>> gd_putfile('blue02.iridis.soton.ac.uk',
   '/home/eres/Soton/Geodise/Geodise-0p3p3/fluent_run.jou',
   '/scratch/eres/fluent_run.jou')
```

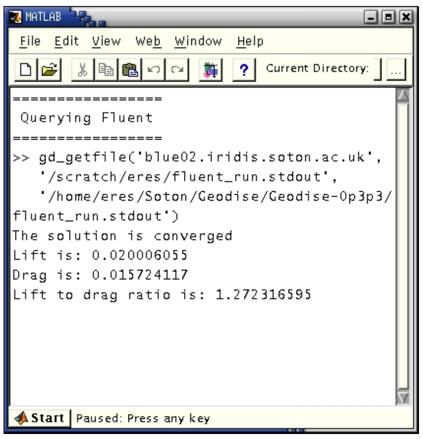
Fig. 2. Generating a Fluent journal file in Matlab environment, and transferring it to the Globus server. Here, the user inputs are the inlet velocity and the angle of attack

A properly generated mesh file is required by the analysis tool, and here the user must generate the mesh file by submitting the geometry to the Gambit mesh generation tool. The user then waits 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 analysis. Therefore, before running the analysis tool the state of the mesh generation and mesh quality are checked by transferring the standard error file produced by Gambit to the local file system and parsing the mesh quality information from that file. If the mesh generation step is satisfactory, the user can now submit the analysis job to the 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. Fig. 3 show the Matlab environment during these steps.



```
====================================
 Running Fluent on the Globus server
====================================
>> gd_jobsubmit(fluent_rsl, 'blue02.iridis.
soton.ac.uk')
GRAM Job submission successful
fluent_handle =
https://blue02.iridis.soton.ac.uk:60291/247
89/1040205524/
>> gd_jobpoll(fluent_handle, 20)
ACTIVE
ACTIVE
ACTIVE
ACTIVE
DONE
```

(a)



```
================
 Querying Fluent
================
>> gd_getfile('blue02.iridis.soton.ac.uk',
   '/scratch/eres/fluent_run.stdout',
   '/home/eres/Soton/Geodise/Geodise-0p3p3/
fluent_run.stdout')
The solution is converged
Lift is: 0.020006055
Drag is: 0.015724117
Lift to drag ratio is: 1.272316595
```

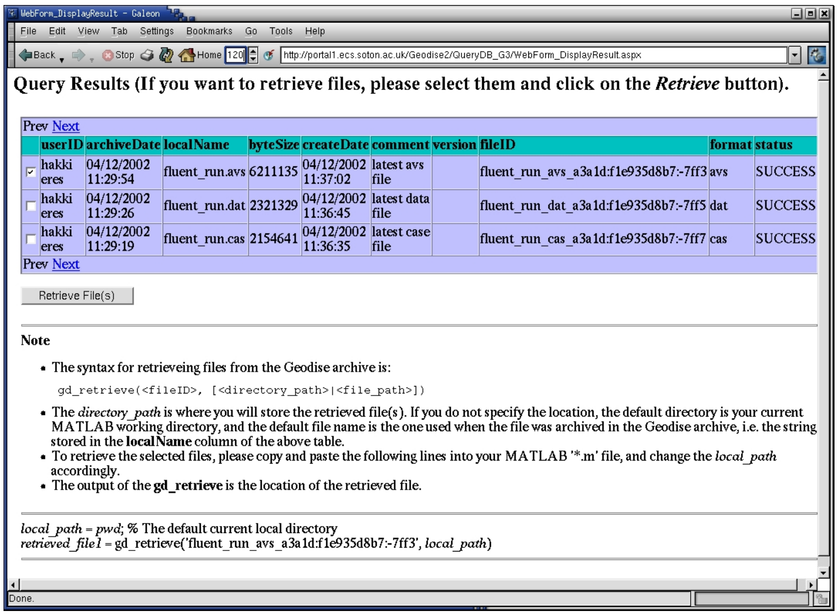
(b)

Fig. 3. (a) Running Fluent on the Globus server by using a proper RSL string and previously generated journal file. The job status is polled every 20 seconds. (b) Transferring Fluent output file to the local file system, and parsing it to retrieve objective function values

Throughout these processes intermediate and solution files are archived in the Geodise repository, with the `gd_archive` command. By associating metadata with the files the design archive may be accessed interactively when required using `gd_query` command which runs a Web browser from the Matlab environment and gives access to archived files (see Fig. 4). Furthermore, the user could be notified about the progress of their simulation by text messages through a SMS. At the end of the simulation an e-mail message containing simulation results is also sent to the user. Ultimately this process would form one loop in an optimisation process.

4 Conclusions and Future Work

The Matlab environment along with the Java CoG [13] provides a flexible and robust user interface for Grid computing. By exposing the compute, data and notification features as toolkit components we are able to construct high level functions which utilise Grid resources for CFD and design search tasks. Given


WebForm_DisplayResult - Galeon
File Edit View Tab Settings Bookmarks Go Tools Help
Back Stop Home 120
http://portal1.ecs.soton.ac.uk/Geodise2/QueryDB_G3/WebForm_DisplayResult.aspx
Done.

Query Results (If you want to retrieve files, please select them and click on the *Retrieve* button).

Prev Next

	userID	archiveDate	localName	byteSize	createDate	comment	version	fileID	format	status
✓	hakki eres	04/12/2002 11:29:54	fluent_run.avs	6211135	04/12/2002 11:37:02	latest avs file		fluent_run_avs_a3a1d:f1e935d8b7:-7ff3	avs	SUCCESS
☐	hakki eres	04/12/2002 11:29:26	fluent_run.dat	2321329	04/12/2002 11:36:45	latest data file		fluent_run_dat_a3a1d:f1e935d8b7:-7ff5	dat	SUCCESS
☐	hakki eres	04/12/2002 11:29:19	fluent_run.cas	2154641	04/12/2002 11:36:35	latest case file		fluent_run_cas_a3a1d:f1e935d8b7:-7ff7	cas	SUCCESS

Prev Next

Retrieve File(s)

Note

- The syntax for retrieveing files from the Geodise archive is:

```
gd_retrieve(<fileID>, [<directory_path>|<file_path>])
```

- The *directory_path* is where you will store the retrieved file(s). If you do not specify the location, the default directory is your current MATLAB working directory, and the default file name is the one used when the file was archived in the Geodise archive, i.e. the string stored in the **localName** column of the above table.
- To retrieve the selected files, please copy and paste the following lines into your MATLAB '*.m' file, and change the *local_path* accordingly.
- The output of the **gd_retrieve** is the location of the retrieved file.

local_path = *pwd*; % The default current local directory
retrieved_file1 = gd_retrieve('fluent_run_avs_a3a1d:f1e935d8b7:-7ff3', *local_path*)

Fig. 4. The Web interface of Geodise toolkit showing query results

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 codes, the integration of various optimisation algorithms, and amalgamation of intelligent workflow composition and retrieval mechanisms to the Geodise PSE. Refining the existing compute components will involve adding client side tools to allow the user to discover compute resources. 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. 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 to be able to extend the lifetime, or perform clean up tasks. The ability to send text messages from mobile devices to the Geodise PSE in order to control processes will also be implemented. In the near future the Geodise toolkit may also exploit the feature of sending binary images to third-generation mobile devices.

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

Acknowledgements

This work is supported by UK EPSRC GR/R67705/01. The authors gratefully acknowledge many helpful discussions with the Geodise team, and researchers from the myGrid team (UK EPSRC GR/R67743/01). We thank Fluent, Microsoft, Level 9 Networks, Epistemics, and Intel for ongoing support.

References

1. Siddall, J.N.: Optimal Engineering Design: Principles and Applications. Marcel Dekker, Inc., New York and Basel (1982)
2. The Geodise Project. (2002) `http://www.geodise.org/`
3. Walker, D.W., Li, M., Rana, O.F., Shields, M.S., Huang, Y.: The software architecture of a distributed problem-solving environment. Concurrency: Practice and Experience **12(15)** (2000) 1455–1480 `http://www.cs.cf.ac.uk/User/David.W.Walker/papers/psearch01.ps`
4. von Laszewski, G., Foster, I., Gawor, J., Lane, P., Rehn, N., Russell, M.: Designing grid-based problem solving environments and portals (2001) `http://www-unix.mcs.anl.gov/~laszewsk/papers/cog-pse-final.ps`
5. Abramson, D., Power, K., Kolter, L.: High performance parametric modelling with Nimrod/G: A killer application for the global grid. In: Proceedings of the International Parallel and Distributed Processing, Cancun, Mexico (2000) 520–528 `http://www.csse.monash.edu.au/~davida/papers/ipdps.ps.Z`
6. Triana. (2002) `http://www.triana.co.uk/`
7. Cox, S.J.: Grid enabled optimisation and design search for engineering (GEODISE). NeSC Workshop on Applications and Testbeds on the Grid (2002)
8. Matlab 6.5. (2002) `http://www.mathworks.com/`
9. Casanova, H., Dongarra, J.: Netsolve: A network-enabled server for solving computational science problems. The International Journal of Supercomputer Applications and High Performance Computing (2000) `http://citeseer.nj.nec.com/casanova00netsolve.html`
10. The Globus Project. (2002) `http://www.globus.org/`
11. Global Grid Forum. (2002) `http://www.gridforum.org/`
12. Commodity Grid Kits. (2002) `http://www.globus.org/cog/`
13. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A Java commodity grid toolkit. Concurrency: Practice and Experience **13** (2001) `http://www.globus.org/research/papers/vonLaszewski--cog-cpe-final.pdf`
14. Verma, S., Parashar, M., Gawor, J., von Laszewski, G.: Design and implementation of a CORBA commodity grid kit (2002) `http://www.caip.rutgers.edu/TASSL/Papers/corbacog-gcw01.pdf`
15. The Globus Resource Specification Language (RSL) v1.0. (2002) `http://www-fp.globus.org/gram/rsl_spec1.html`
16. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Secure, efficient data transport and replica management for high-performance data-intensive computing. IEEE Mass Storage Conference (2001) `http://www.globus.org/research/papers/msc01.pdf`
17. Abbott, I., von Doenhoff, A.: Theory of Wing Sections. Dover Publications, New York (1959)
18. Fluent. (2002) `http://www.fluent.com/`