

Improving Usability in a Grid-Enabled Environment

Agah Ejura Ogungboye

Master of Philosophy(MPhil)

University of York

Department of Computer Science

June 2010

Abstract

Several scientific application developers, especially intermediate users (typified by not wanting to learn any special program to use the Grid but have some average knowledge of computers) would want to exploit the possibilities offered by Grid computing. These developers have their scientific agendas to pursue and lack the time or skills to explore the vast wealth of Grid technologies. Presumably, they would rather be presented with a simple system that resembles the scientific programming paradigm that they are familiar with.

This project is aimed at designing and developing a more user friendly interface to enable scientific application developers to use networked computer systems (Grids) simply. However, the author investigated how other source codes would compile remotely and how to deal with compilers of different versions of Operating system. This project aimed to show that it is possible to reduce the steps taken by a user to grid-enable an application by designing a Grid user interface that can abstract the complexity that is involved with achieving this process. It also aimed at showing that submitting jobs on a grid can be done efficiently and extended to a wider range of users.

Building this user interface necessitates investigating the use of new and pre-existing software modules set up as Web services - a current challenge in this research area. The system that was developed investigated current user applications (and their designs); especially how a new user interface would allow such an application to run on a Grid via two approaches. These include the transfer of compiled code and executing it on the target computer; and transfer of uncompiled code, its compilation and executing it on the target computer. Both approaches meant abstracting the problems associated with moving the software and running it on one of the machines in a Grid.

The author encountered software library challenges during the course of the project; it became apparent that posed a problem for uncompiled and compiled code. The challenges include the type of compiler, the version of compiler, the type of machine

compiled and external dependencies needed by the uncompiled/compiled code to run successfully. However task for executing the uncompiled code proved simpler than the compiled code. A simple C++ program was used to test this. Issues arising from this test were also explored. A detailed analysis has been made on how to verify dependent libraries on a code, transferring the code to the Grid node and then running the code successfully.

Different group of users consisting of six (6) domain expert users and a usability expert evaluated the graphical interface built. The domain experts helped in determining the effectiveness of the system and the usability experts' helped in the usefulness of the interface. The methodology used was heuristic evaluation. Their interaction with the interface, results and how it can be improved is documented in chapter 5 of the thesis.

Contents

ABSTRACT.....	II
CONTENTS.....	IV
LIST OF FIGURES, LISTINGS AND TABLES.....	VII
ACKNOWLEDGEMENT.....	VIII
DEDICATION.....	IX
AUTHORS DECLARATION.....	X
1 INTRODUCTION.....	11
1.1 Thesis Hypothesis	13
1.2 Thesis Motivation	14
1.3 Thesis Objectives	17
1.4 Thesis Methodology.....	18
2 LITERATURE REVIEW AND SURVEY	20
2.1 Background on the Grid.....	20
2.1.1 History of the Grid	22
2.1.2 Taxonomy of Grid Applications	25
2.1.3 Grid Components and Technologies (OGSA,Web services, Workflow)	26
2.1.3.1 The Globus Toolkit (Evolving Stages)	27
2.1.3.2 The Versions of the Globus Toolkit.....	27
2.1.4 Methods for Building Grid Applications	33
2.1.5 Grid WorkFlow Projects	34
2.1.5.1 MyGrid Project	34
2.1.5.2 Triana	35
2.1.5.3 Geodise	37
2.1.5.4 DAME.....	38
2.1.5.5 Summary of Grid Workflow Projects reviewed	39
2.2 Usability.....	40
2.2.1 Usability principles and guidelines.....	40
2.2.2 The Classic Software life Cycle: The Waterfall Model.....	41
2.2.3 Iterative Design and Prototyping	42
2.2.4 User-Centred Design.....	43
2.2.4.1 Usability Engineering	44
2.2.5 Design Rationale	45
2.3 Evaluation criteria Technique	46
2.3.1 Evaluation through expert analysis	46
2.3.2 Evaluation through user participation.....	49
2.3.3 Establishing evaluation criteria.....	51
2.4 Previous Research on Usability of the Grid.....	52
2.4.1 Grid Middleware aimed at Grid Usability	52
2.4.1.1 The GridLab Project	52
2.4.1.2 GridGain	55
2.4.2 User-friendly Grid projects	56
2.4.2.1 The eMineral Project.....	56
2.4.2.2 Ganga, A user interface for Atlas and LHCb	58
2.4.2.3 Styx Grid Services (SGS)	62

2.5	Requirements derived from Existing projects.....	63
2.6	Conclusion	66
3	SYSTEM DESIGN AND IMPLEMENTATION.....	67
3.1	Design methodology for the project	68
3.2	Exploring the user Community	69
3.2.1	Issues in Usability evaluation of the Grid System	69
3.2.1.1	Consider Users Types	69
3.2.1.2	Task analysis	70
3.3	Determining User Requirements.....	70
3.3.1	Use Case for User1	71
3.3.2	Use Case for User2	72
3.3.3	Use Case for User3	72
3.3.4	Users experiences in using the Grid.....	73
3.3.5	Requirements for the proposed system	74
3.4	Using the Grid.....	74
3.4.1	Proposed way of using the Grid.....	75
3.5	Grid-enabling an application.....	75
3.5.1	Grid Interface Architecture	75
3.5.2	Job Description	75
3.5.3	Grid interface checks if request can be handled	76
3.5.4	The Request is verified	78
3.5.5	Job is submitted to the underlying Grid environment.....	78
3.5.5.1	Application Execution	79
3.5.5.2	Scheduling Job Execution.....	80
3.5.5.3	Application Specific requirements.....	81
3.5.6	Job Monitoring.....	81
3.5.7	Result Returned to user	82
3.6	Deployment.....	82
3.6.1	Different ways for Application-based deployment.....	83
3.6.1.1	Copy Libraries to Resource provider	83
3.6.1.2	Copy Libraries to the user remote home drive.....	83
3.6.1.3	Virtual Machine Deployment	84
3.6.1.4	Java virtual machine Deployment.....	84
3.7	Issues in implementing the interface.....	85
3.7.1	Computer Architecture and Design	85
3.7.1.1	Data and program representation	85
3.7.2	Library issues	87
3.7.2.1	Linkers and Loaders.....	87
3.7.2.2	Types of Program Libraries (x86 architecture Linux)	89
3.7.2.3	How libraries are used (x86 architecture Linux)	91
3.7.2.4	Deploying an Application	91
3.7.2.5	Running the executable from the remote end	92
3.7.2.6	The Library verification process	94
3.7.2.7	Running sample executable from remote end.....	100
3.7.3	Data and Application	101
3.8	Thesis Constraints	103
3.9	Conclusion	104
4	GRAPHICAL GRID USER INTERFACE	105

4.1	Tools for Implementing the Grid User Interface	105
4.2	User Centred Design process	106
4.3	User Interface Window components.....	107
4.3.1	My Workspace Window	107
4.3.2	Services/Applications Window.....	107
4.3.3	Available Grids Window	108
4.3.4	Data Repository Window.....	108
4.4	Interface Diagram	108
4.5	Setting up the environment	111
4.6	The different classes of the interface	113
4.6.1	QMainWindow Class.....	113
4.6.2	MyWorkspace Class	113
4.6.3	Application Class	113
4.6.4	Data Repository Class.....	114
4.6.5	Services Class	114
4.6.6	The Class Diagram of the interface.	114
4.7	The Grid User Interface Software Setup.....	114
4.8	Job submission Process.....	115
4.9	Conclusion	117
5	EVALUATION	118
5.1	Conducting the evaluation	119
5.1.1	Use Case for Evaluation.....	119
5.1.2	Results from three domain experts (Questionnaire and think aloud)	120
5.1.3	Results from the Usability Expert (Heuristic Evaluation)	122
5.1.4	Results from three domain Experts (Work-Domain Expert Analysis)	124
5.1.5	Validity of the Results.	126
5.1.6	Usability problem of the interface	127
5.1.7	Requirements specification for the Users	128
5.2	Comparison of Grid User interface with other approaches.	128
5.3	Conclusions.....	129
6	CONCLUSIONS AND FURTHER WORK.....	131
6.1	Conclusions from experiences	131
6.2	Further work.....	132
6.2.1	The Grid User Interface	133
APPENDICES.....		135
Appendix A: DAME: A scientific developer experience of building a Grid environment.		135
Appendix B: Conferences and Seminar		143
Appendix C: Setting up the Grid User Interface.....		146
Appendix D: Screen shots of Grid User Interface		152
Appendix E: User Interface Evaluation sheet.		160
Appendix F: Questionnaire Sheet.		162
DEFINITIONS		166
GLOSSARY.....		167
REFERENCES.....		168

List of Figures, Listings and Tables

Figures

Fig 2.1: Position of Grid middleware in a Grid environment.....	21
Fig 2.2: The GT1-Job submission process.....	30
Fig 2.3: The GT2-Job submission process.....	31
Fig 2.4: The GT3-Job submission process.....	32
Fig 2.5: A snapshot of Triana user interface.....	37
Fig 2.6: Screen shot of the e-Mineral web based tool.....	57
Fig 3.1: Grid Interface architecture.....	76
Fig 3.2: Flowchart of job submission process.....	94
Fig 3.3: Flowchart of library verification process.....	96
Fig 3.4: Detailed flowchart of library verification process.....	98
Fig 4.1: The interface model.....	110
Fig 4.2: The workbench of the Grid user interface.....	112
Fig 4.3: The class diagram.....	116

Listings

Listing 3.1: Determining the endianness of a computer.....	86
Listing 3.2: Illustrating header files.....	88
Listing 3.3: Updating the LD-LIBRARY PATH.....	95

Tables

Table 2.1: Comparison of two approaches to implementing a Grid interface.....	34
Table 2.2: Summary of Grid Projects reviewed.....	64
Table 3.1: Sample table information about processing nodes.....	77
Table 3.2: Comparison of static and dynamic linking.....	92
Table 5.1: Users' response to questionnaire.....	121
Table 5.2: Heuristic evaluation sheet 1(Usability expert).....	123
Table 5.3: Heuristic evaluation sheet 2(Work domain expert).....	125
Table 5.4: Group based domain evaluation.....	128
Table 5.5: Users' requirements review on evaluation.....	129

Acknowledgment

I would like to acknowledge the one person who helped towards the successful completion of this report, my project supervisor Prof Jim Austin for his invaluable support and advice through the duration of my project. He had faith in me and that kept me going. I also want to thank Professor Richard Paige for his ever present help. I would like to acknowledge Aaron Turner who always gave me the technical support I needed during the project. My appreciation goes to Boijan Liang, Mark Jessop, Martyn Fletcher who assisted me in one way or the other.

I owe the commonwealth scholarship for giving me the opportunity to undergo this program and all their moral support when the program was extended due to circumstances beyond my control.

My gratitude goes to my husband Mr Razaq Muyiwa Ogungboye for being steady and encouraging of this program and giving me his full support both financially and emotionally.

My thanks goes to my sister Igbi Ojoma Idachaba and my mother-in-law Alhaja HS Ogungboye who dedicated a lot of their time just to make sure I finish this program. I also want to thank my Friends Iyabo Umar-lawal and Dr Diana Gimba for their moral and emotional support. I would also like to acknowledge the following people for just being there: Professor J.N Egila, Mrs Unoko Okonkwo, Mr Kunle Ibrahim, Mr Makoji Aduku, Mrs Acharu Aweto, Mrs Iganya Okeme, Miss Chogu Idachaba, Miss Ileanwa Idachaba, Miss Uyo-Ojo Idachaba.

I would like to express my love to my constant companion Miss Folashade Eke Ogungboye whose company and presence remains a motivating factor. I also would like to acknowledge my mother Mrs Aye H Idachaba for her prayers and love.

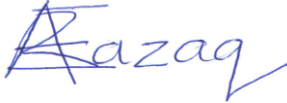
Dedication

This report is dedicated to my Husband Mr Muyiwa Razaq Ogungboye who has remained my confidant and my late father Mr Emmanuel Mamudu Idachaba.

Author's declaration

All research presented in this thesis was initiated and conducted by the author. The author is responsible for the research presented in this thesis.

Signed



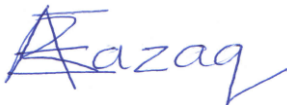
Agah Ejura Ogungboye

Date...25-February-2013

Statement.

This thesis is the result of my own investigations, except where other stated. Other sources are acknowledged by references. A bibliography is appended.

Signed



Agah Ejura Ogungboye

Date...25-February-2013

1 Introduction

The word “Grid”, when applied to computing systems, is defined analogously to the national power grid that provides continuous access to electrical power to individuals and organizations. Most homes in every part of the world are provided with electricity, and its users are abstracted from concerns as to where the power comes from. This is the main idea behind the Grid: to connect multiple regional and national computational grids to create a universal source of computing power (Foster, Kesselman 1999). The Grid is a system that is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports a collection of users and resources (virtual organizations) across traditional administrative and organizational domains (real organizations) [82]. A Computational Grid is therefore defined as a computing infrastructure that enables the virtualisation of distributed computing and data resources such as network bandwidth, storage capacity and processing power to create a single system, granting users and applications access to vast IT capabilities for solving problems too intensive for a standalone application.

The Grid is born out of specific and real problems and there are technologies that address these problems. A specific problem can be found in the DAME project [60] where aircraft companies needed to study and analyse the volumes of engine data that was generated each time that a plane landed. The data generated could be in terabytes hence the need arose to build a system that could handle and collate this volume of data simultaneously from different locations. This is where the Grid is most applicable - for coordinating resources that are not subject to centralised control (Foster 2002). In the trend of the computing world, every user expects to submit their executable (job) and have it executed as fast as possible. Scientific application developers’ deal with applications that generate large amount of data and it is becoming increasingly difficult for one super computer to handle such data. This highlights the need to share computing power, storage and memory amongst different supercomputers located in different locations. In addition, there is also the need to lower the cost of computing by the collection of spare cycles to form a logical processing node. Grid is the most promising solution to the above-listed data

management challenges. The Grid offers the promise of access to increased computational capacity by coordinating the simultaneous use of computers linked by networks (Foster, Kesselman 1999). Through this technique, the “size” of an application can be extended beyond the resources available at a particular location – indeed, potentially beyond the capacity of any one computing site anywhere (Foster, Kesselman 1999). Consequently many scientific application developers would like to make use of the existing possibilities opened up by the advent of the Grid and would like to be presented with a system close to the programming paradigm they are used to. (Chin J et al 2004).

The Grid technology is a popular research area in distributed computing that has been embraced by various organisations. Most Grid projects are carried out by the Academic sector in collaboration with major players like Microsoft, IBM and Sun. The main body Open Grid Forum (OGF) that is responsible for setting standards for this technology has initiated many conferences to achieve this aim. There have been a lot of Grid projects in the last few years both nationally and internationally. Some of these projects including Triana [97], MyGrid [77], CARMEN [55], DAME [60], eMineral[62], and others mentioned in the literature review of this thesis are scientific in nature. Triana [97] and MyGrid [77] are built using workflows as a means to connect the data resources and processing power needed to describe a Job before it is ready for submission to the Grid.

This thesis aims to address the complexities that are involved in building a Grid network. These complexities arise due to incompatibility with the distributed compute and data resources and the applications that access the Grid. Some of these complexities which the thesis aim to address include; determining a global way to monitor each local domain that is part of the grid, incompatibility with legacy codes and accommodating different user application that would like to make use of the Grid. In this project, we explored building an interface that can accommodate various legacy codes. Further work was done on C++ source code and compiled code and how these can be integrated successfully with an existing Grid environment with focus on the usability aspects of the interface that was built. This interface can allow a user to submit serial C++ source or compiled code, have their libraries correctly

verified, monitor their jobs, and have their result displayed on successful job completion. This interface can be extended to handle parallel code in further work.

The thesis also identified users according to their level of knowledge about computing. The three groups identified are the Basic, Intermediate and Expert. The Basic user describes an everyday user who is concerned about an interface that is simple for him to submit his job, pick his data and have the result returned to him. The intermediate user is a basic user but with some moderate knowledge of computing. They can write/modify scripts and may want to have control over their job and even link processes into workflow and are typified of not wanting to learn any special programming language to use the Grid. The expert user is very advanced with the computing technology and understands the concept of the Grid and what it can do for him. He can even write his own processes using message passing interface (MPI) (Crichlow 2000) a popular parallel programming language. This user is familiar with the tools needed to build the Grid and would not even mind learning any language that he needs to use the Grid. The group of users targeted in this thesis are the intermediate group of users.

1.1 Thesis Hypothesis

Scientific Application developers write complex programs; these programs normally benefit from parallelization to improve performance. This has brought about different methods and ways to handle such code and its optimizations, and different technologies under the broad heading of distributed computing have emerged from this development. These include CORBA [58], DCE (Millikin, 1994), and Web services amongst others. However these technologies are insufficient for application developers who generate large volumes of data and need more processing nodes to make their programs more efficient.

The Grid technology provides the platform and infrastructure for performing such high computational tasks and even storage of large volumes of data that can result from such tasks. However the Grid technology with all its benefits introduces new challenges; a new programming model is needed in some cases, incompatibility with

legacy codes and some Grid environment require re-writing codes for a successful execution of existing codes on them. Another challenge is usability as this deals with how effectively people can interact with the Grid (Chin J et al 2004).

Current usability of the Grid is still time consuming for the intended users who have to learn a complete new environment and a different way to interact with the environment to get their application grid-enabled. Users usually struggle with understanding the functionality offered by these Grid environments as the interfaces are usually very cumbersome. This thesis is therefore concerned with building a graphical interface that requires minimal operation effort therefore taking advantage of the potentials offered by the Grid. We built a new interface that will allow users a simple and straight forward means of submitting jobs and integrating their existing codes to the Grid environment.

The hypothesis of this thesis is then based on the fact that it is possible to improve the usability of the grid for intermediate scientific users by building a new type of graphical user interface tool that would allow users to submit and monitor their jobs on the underlying grid environment.

1.2 Thesis Motivation

There have been a lot of Grid projects that have shown the usefulness of the Grid especially in scientific applications and these projects have had to deal with the fast and emerging nature of the Grid technology. These projects have mostly concentrated on the functionality of the system rather than the usability. For this reason they are not widely accepted by the larger communities who just want a simple Grid system they can work with (Chin J et al 2004). To illustrate this point, the DAME project is looked as a case study to investigate how scientific developers try to build Grid applications/environments today. A scientific developer's (Jessop, Appendix A) experience and views in building a particular Grid project DAME was sought and is summarised in the next paragraph.

DAME (Distributed Aircraft Maintenance Environment) project was an eScience project that was meant to demonstrate the usefulness of the Grid in engineering

applications. DAME aimed to build an end workbench for; data collection and management, data processing, engineering tools, result correlation and knowledge capture. Several Grid tools were built to achieve a working DAME system namely SRB, AURA, SDE and PMC. The Storage Resource Broker (SRB) provided the data storage element [91]. It is a client server system for joining together distributed storage resources into a virtualised storage system and was installed in all the data centres. The data search was implemented using the AURA (Advanced Uncertain Reasoning Architecture) technology. AURA is a generic family of techniques and implementation intended for high-speed approximate search and match operations on large unstructured datasets. The technology is based on high performance binary neural network called Correlation Matrix Memory (CMM) and several CMM elements are used in combination to solve soft or fuzzy pattern-matching problems [54].

A Signal Data Explorer (SDE) tool was developed for exploring and searching time series data. A distributed search process; Pattern Match Controller (PMC) was built to manage the large volume of data with the SDE as a front end. The PMC deployed a Grid service at each data centre coupled with a local search engine, which worked on data stored in the local repository. The SDE connected to a single PMC Grid service requested for a distributed data to be searched. The PMC then connects with the other sites to search the data. All data access was via the SRB. Results could be obtained through the SRB.

This feature demonstrated the power of the SRB as users saw data as part of a single repository. Globus [92] was the chosen middleware to implement the processing Grid infrastructure. Globus was hard to use as previous versions did not do what they were supposed to do and the documentation was almost non-existent. All versions of Globus toolkit from GT1 to GT4 were explored during the course of this project. Services deployed with previous versions were incompatible with the new one and this was a drawback. The OGSA architecture is heavily layered and particularly difficult to work with.

The development process of the DAME project was difficult and time consuming mainly due to the lack of documentation and pace of change associated with the

middleware (Globus Toolkit) necessary to build the Grid network. Building this environment required an in-depth knowledge of the Globus toolkit. The debugging and deployment stage required a great level of understanding even for experienced application developers.

The SDE which was the front-end for visualising and browsing complex signals amongst other functions was a very complex tool to work with. I personally had some interaction with the User interface and needed a lot of time and effort to fully understand what it was trying to achieve. I believe that potential users of this tool would struggle with initial interaction with this tool.

Some initiatives to improve this situation were conceived by the OMII project. The Open Middleware Infrastructure Institute (OMII) was an initiative to preserve and consolidate the achievements of the UK e-Science programme by collecting maintaining and improving software modules that form the key components of a generic Grid middleware (Atkinson et al 2004). Their goals and aims were quite impressive because they wanted to improve the efficiency of e-Science research projects, increasing the level of software reuse between software projects and thereby to achieve a better utilisation of development resources. But unfortunately this still proved very difficult to use as a lot of coding and understanding of this middleware is required to correctly program the Grid. More effort needs to be put in the usability aspects of the Grid as this is the front end that is presented to the user for his access to the Grid. The users need to focus more on how to use the system to achieve the end goal than spending time on knowing how the system works (Pancake, 2003).

It is therefore hoped that improving the usability of the Grid would make it more accessible to the wider community who are more of the intermediate users. This thesis aimed to build a graphical user interface that would hide the complexities that are involved in using the Grid to submit jobs.

CARMEN users are a typical user community for this project. These users are neuroscientists that want to collect, visualise, store, annotate and analyse data. The CARMEN Project [55] aims to use the Grid to allow users to archive these datasets in a way that makes it accessible for other developers to exploit. This MPhil project has

also looked at the usability aspects of the CARMEN project. The Carmen project started in the year 2007 and as such getting the necessary information to help build the graphical user interface have not been readily available.

Our test group users in the project are a collection of people from my group (Advanced Computer Architecture Group) who have a fair knowledge of the Grid and can help to perform a heuristic evaluation of the Grid User Interface. Some of these users consist of six scientific application developers of the Grid.

The interface uses a simple C++ Job that can be submitted either as a source code or as a compiled code and the evaluation was based on the steps taken to submit the job and how easy it was to use the interface for job submission. It is hoped that this interface can serve as a basis for further research work that can be tailored to a specific user domain with specific ways of submitting a job using a specific programming paradigm.

1.3 Thesis Objectives

Based on the usability studies carried out at the beginning of the project and from initial requirements, the thesis aimed to achieve the following main objectives:

1. Design of a tool that allows a user submit a job more effectively than before.
2. Improve the effectiveness of the steps taken to Grid-enable an application.

We then believe that these objectives will be able to answer the following list of research questions:

- Can we build an interface that accommodates user's legacy codes in an efficient manner?
- Can we build an interface that requires minimal effort and knowledge to use?
- Can the interface accommodate the intermediate group of users (with good knowledge of computing but inexperienced with the Grid)?
- Can we handle the successful verification of libraries and tools at the remote end of a Grid environment that will facilitate the successful execution of the user's application? (The underlying system would accommodate compiled and

uncompiled codes together with the necessary system libraries needed to get them compiled and executed successfully)

- Can users effectively monitor and get useful feedbacks from the interface about their work? (The interface would give the user exact information on actions they perform from the submission of their tasks to the successful execution of their tasks including any errors they may encounter in the process)

1.4 Thesis Methodology

A review of past Grid projects looked at the two ways of deploying Grid applications. These are application based and portal based (Section 2.1.4). This project has identified the type of users to be intermediate users whose initial expectation of the system was identified from the initial requirements. The application-based approach involved building a graphical Grid user interface for allowing users to effectively compile and/or run their applications on the Grid. Presently these users access the Grid via the command line interface to submit and monitor their executed applications. The portal approach is not ideal for these users as it restricts them to a specific domain where it is difficult to integrate their existing applications. This is because they want to use a Grid system that maintains the conceptual model of their original system and therefore still have a complete interaction with their desktop environment.

This project is designed with specific users in mind, as this is needed to determine the requirements and capabilities but would have the potential to cover an extremely large number of users and application domain. The stakeholders are scientists who would like to become users of the grid and the application domain that was used to determine these requirements are the CARMEN users but they would not be the users that would eventually test this interface. A typical user group community was looked at; three different grid users were interviewed and their input has helped in understanding the way people currently use the Grid to submit jobs

The Grid user interface initially was a high level design, which gave a brief overview of the functionality of the system. A semi functional prototype version of this interface has been developed in this research. Using this semi-functional prototype,

volunteer users consisting of usability experts and domain experts have carried out evaluation of the system to determine if the basic requirements of the system have been met. The domain experts consisting of the ACAG (Advance Computer Architectural Group at York) evaluated the system and answered questionnaires individually and as a group. The usability expert performed a heuristic evaluation of the system to determine if the usability heuristics have been met. This evaluation is discussed in a later chapter.

2 Literature Review and Survey

2.1 Background on the Grid

The main idea behind the Grid is to connect multiple regional and national computational Grids in order to create a universal source of computing power (Foster, Kesselman 1999).

The Grid is therefore defined according to Foster (Foster, 2002) using the following checklist:

- (1) coordinates resources that are not subject to centralized control....
- (2)using standard, open, general-purpose protocols and interfaces.....
- (3)to deliver nontrivial quality of service

The Grid provides standard access to a range of e-services including knowledge, information, computational, experimental and data storage systems (Allan et. al. 2000). The Grid is usually distinguished between computational Grid and the more enhanced Data Grid and there are additional classifications such as:

- Enterprise Grid: This is loosely defined as a distributed system that aims to dynamically aggregate and co-ordinate various resources across the enterprise and improve their utilisation such that there is an overall increase in productivity (Buyya et al 2005).
- Collaboration Grid: These Grids found more in the scientific community, involve multiple organizations and individuals, security domains, protocols, discovery mechanisms and heterogeneous hardware, collaborating to share their resources to make the most effective use of it for their combined communities [77]. A very good example of a collaboration Grid is the White Rose Grid [77].
- Cluster Grid: This is typically a set of homogenous systems that are tightly coupled in a dedicated network and are able to serve as excellent Grid nodes that can for example be assigned MPI (message passing interface) jobs to

perform [57]. They are aimed at high performance/throughput computing and are mostly, workload scheduling systems found in both commercial and eScience environments [82].

A very important aspect of the Grid, termed the ‘middleware’ is responsible for the coordination of geographically dispersed resources including humans (see figure 2.0). A Grid middleware can be defined as the “the services needed to support a common set of applications in a distributed network environment” (Aiken et al 2000). There exist software technologies that are already being used by different groups involved in Grid projects. These technologies form a major part of the Grid Middleware. Examples of this include the Globus [69], Jini [73], OMII [83] and E-legion [63]. Globus, quite popular amongst organisations and the academic sectors involved in Grid computing is my choice of Grid middleware and as such discussed in detail.

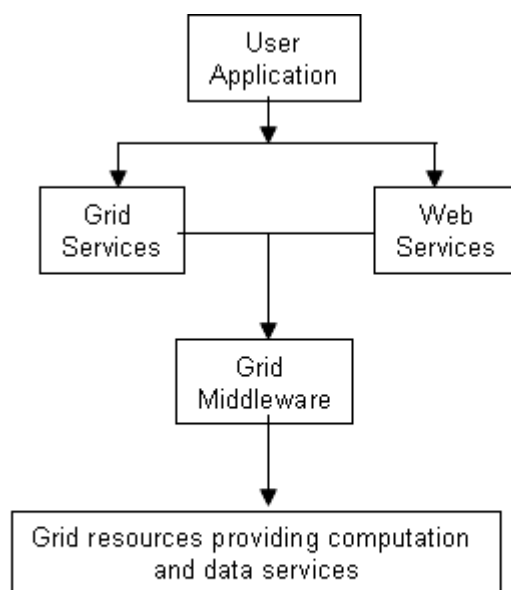


Fig 2.1. Position of Grid middleware in a Grid environment

In the following section a number of areas are considered. The history of the Grid is discussed. The Globus Toolkit middleware is also discussed in detail. In OGSA [65], a Grid service is basically a web service with some additions to make it persistent (it is able to store state information persistently rather transiently at the server beyond the

lifetime of a single request). Finally the current Grid technologies deployed whether Portal based or application based are looked at and compared to highlight the strength and weaknesses of each approach. Figure 2.1 above shows the position of the Grid Middleware in the overall Grid technology from my own point of view.

2.1.1 History of the Grid

The Grid is another research topic under the broad area of distributed computing. The term was introduced in 1998 with the launch of the book “The Grid: .Blueprint for a new computing infrastructure”. Before the advent of the Grid other distributed technologies existed. To further understand this section it would useful to define some terms relating to all aspects of distributed processing:

- Computer Programs (Jobs) are instructions for a computer and are either executable programs or a source code from which executables are derived.
- A Process (task) is an instance of a computer program that is being sequentially executed by a computer system that has the ability to run several computer programs concurrently.
- Central Processing unit (CPU) or a processor is a description of a class of logic machines that can execute computer programs.
- Multitasking is a method by which multiple tasks also known as processes share common processing resources such as a CPU.
- Parallel processing is the simultaneous use of more than one CPU to execute a computer program. In practice it is difficult to divide a program in such a way that separate CPUs can execute different portions of the program without interfering with each other. Most computers have one CPU and others have more than one CPU. Those with one CPU can handle parallel processing by connecting the computers in the network and this requires very sophisticated software called Distributed processing software.
- The Grid is a very sophisticated extension of parallel processing which extends it beyond just the simultaneous execution of different parts of a computer program by different CPUs. The Grid offers other value added services like authentication, Data management, information management, service discovery, Interactive feedbacks and so forth.

- Message Passing interface (MPI) is a low-level programming model for developing applications on networks of workstations and massively parallel machines. MPI facilitates the programming of parallel and distributed applications primarily in C and Fortran 77 (Crichlow, 2000). It has also been used in MPICH-G2, a Grid-enabled implementation of MPI that allows a user to run MPI programs across multiple computers at the same or different sites using same commands used on a parallel computer (Foster, 2003).

The history of the Grid dates back to the year 1995 when the I-way project (DeFanti et al 1996) was launched in respect to metacomputing. New classes of high-performance application were being developed that required more than a single computer for a successful and efficient execution. Metacomputing is the field of computing focused on the methodological and technological aspects of the development of large computer networks/Grids such as the internet and other territorially distributed computer networks for special purposes. This was an improvement on existing technologies like parallel and distributed computing. The IWAY (Information Wide Area Year) was a yearlong effort to link national test beds based on ATM (asynchronous transfer mode) to interconnect supercomputer centres, virtual reality (VR) research location and applications development site (DeFanti et al 1996). The networking experiment connected super computers and other resources at 17 different sites across North America and had 60 groups develop applications in areas of large-scale scientific simulation. The project major goal was to work with applications community to explore the benefits of distributed supercomputing. Applications that would use more than one supercomputer and one or more VR devices to explore collaborative technologies were also explored. The problems that prevented the widespread use of the distributed computing over ATM networks were investigated and these areas were identified; security, uniform computing environments, wide area scheduling and resource reservation, and distributed collaborative VR. According to (DeFanti et al 1996) the major contribution of the I-way project which finally gave birth to GRID computing are:

- Motivated applications groups to consider using VR and supercomputing together.

- Established a framework for building national research infrastructure in partnership with carriers' vendors and applications community.
- Built the first nationwide infrastructure to support collaborative computational science projects on a large scale.
- Developed a prototype software environment and wide area scheduler to enable easy use of distributed resources without knowledge of their location and configuration and with a high degree of security.

The I-way project ended in 1996 but the idea of the Grid had already been borne and with it was the creation of Globus, a metacomputing infrastructure toolkit. The heterogeneous and dynamic nature of metacomputing systems limited the applicability of the parallel computing tools and techniques that existed at that time. There was a requirement for advances in mechanism tools and techniques in this area and the Globus project was created to accelerate this advances. The focus on Globus at that time was to develop low-level mechanisms that can be used to implement higher-level services and techniques that allow those services to observe and guide the operation of these mechanisms (Foster I et al 2006). The Globus Toolkit to date is discussed in section 2.1.3.1.

Open Grid Services Architecture (OGSA) is an idea behind how Grid technologies could be aligned with existing web service technologies to capitalise on desirable web services. The OGSA leveraged on the experiences gained with the Globus Toolkit to define conventions and WSDL interfaces for a Grid Service. A Grid Service is a Web service that conforms to a set of conventions (interfaces and behaviours) that define how a client interacts with a Grid Service (Foster et al 2002). Open Grid Services Infrastructure (OGSI) building on both web and Grid services technologies defines the mechanisms for creating, managing and exchange of information amongst entities called Grid services. The OGSI defined a model that extends WSDL and XML definition to incorporate the concepts of [68]:

- state Web services,
- extension of Web services interfaces,
- asynchronous notification of state change,

- references to instances of services,
- collections of service instances, and
- service state data that augments the constraint capabilities of XML schema definition.

This then implied that a Grid service instance is a service that conforms to a set of conventions, expressed as WSDL interfaces, extensions and behaviour for such purposes as lifetime management, discovery of characteristic, and notification. Grid services can then provide for the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications [68].

Grid computing, a special form of distributed computing, encompassing preceding distributed technologies handle more complicated execution of processes. Grid computing can therefore be described as a special type of parallel computing which relies on complete computers (with on board CPU, storage, power supply, network interfaces etc) connected to the internet by a conventional network interface like the internet. The Grid then aims to offer the following not offered by other distributed technologies (Foster et al, 2001):

- Single sign on to different services
- Coordinated sharing of resources (Reuse of workflows etc).
- Access to seamless computer power, storage and network interfaces.

2.1.2 Taxonomy of Grid Applications

There are many applications that can benefit from Grid-enabling and these categories of application are discussed below. Grid applications usually follow this order of operation; an executable sometimes called a program runs, takes in input files processes them to produce an output file (which could be another input file). This program is usually referred to on the Grid as a job. Based on this principle, (Suicu et al, 2008) derived taxonomy of Grid applications containing six basic categories:

- SPNF-Single Program No File
- SPSF-Single Program Single File
- SPMF-Single Program Multiple File

- MPNF-Multiple Program No File
- MPSF-Multiple Program Single File
- MPMF-Multiple Program Multiple File.

With this taxonomy, a grid application is in fact a collection of jobs. For example a SPMF grid application may consist of several jobs each performing the same transformation (program) on separate input files (encrypting all the files from a directory for example) (Suciu et al, 2008). The six basic categories has been listed according to the number of jobs each of these scenarios produces with the MPMF type of application producing the highest number of jobs. In this thesis we have worked with SPSF application but it can be extended to handle the SPMF MPNF and MPSF type of applications. Clearly the MPMF type of application exhibits the highest form of parallelism.

2.1.3 Grid Components and Technologies (OGSA, Web services, Workflow)

The OGSA (Open Grid Services Architecture) is a middleware evolution towards a Grid system architecture based on web services concepts and technologies [85]. The OGSi (Open Grid Services Infrastructure) is the base infrastructure on which the OGSA is built. These are all coupled together and known today as the Globus Toolkit (GT). The Globus toolkit [94] is a joint work of Argonne National Laboratory and the University Of Southern California Information Science Institute with other contributors. It is the most widely used by emerging Grid projects. Globus provides a standard set of services for user authentication, resource location, resource allocation, configuration, communication, file access, fault detection and executable management. This helps the application developer and tool builder to tackle the challenges of constructing a Grid-aware engineering and scientific applications. There is currently version GT5. The GT has made tremendous impact to a lot of developers. The Globus toolkit in 2007 was the basis on which most IT companies like IBM built significant commercial Grid products.

The Globus Toolkit is open source and is not platform dependent. It can be modified to further suit the environment on which it runs by the application developer. Some science projects like DAME have used it as their middleware software. It is packaged

as a set of components that can be used either independently or together to develop applications [94]. There has been lots of effort from the Global Grid Forum (GGF) in making the Globus Toolkit a de facto standard for major protocols and services and has acquired a large tool/user base. There are lots of successes and it still has deficiencies but the fact remains that the speed at which these deficiencies are being addressed makes it a toolkit to reckon with. There is currently a finalised version 1.0 of the Open Grid Services Infrastructure (OGSI) [86].

Figure 2.2 gives a brief view of how job requests are made over a Grid system with the OGSA as the middleware. The Globus toolkit has evolved from the GT1, GT2, GT3 and currently GT5. Figure 2.2 is a rough view of how the DAME project [60] views the Globus Toolkit.

2.1.3.1 The Globus Toolkit (Evolving Stages)

The Globus Toolkit is the software implementation of the OGSA and OGSI. The toolkit has been designed to use (primarily) existing components, including vendor-supplied protocols and interfaces (Foster et al, 2001). The implementation of the Globus toolkit as seen by the DAME project is explained briefly. The Grid Security infrastructure (GSI) is responsible for the authentication, communication protection and authorization. Authorisation is via a single sign on using the X.509 certificates. Once a user logs on to the Globus toolkit, he/she gets authenticated on every single processor on the Grid network to provide data and processing power to the Grid. The processors all have a GridMapFile, which is a reference between the user's single sign on, and the machine the user wants to use. The Grid Resource Information Service (GRIS) handles the job submission to the machines. The Globus toolkit has evolved from the GT1 to the GT5. All of these stages are discussed briefly in the following paragraph.

2.1.3.2 The Versions of the Globus Toolkit

The GT1 was the ever first version of the Globus Toolkit. As shown in figure 2.2, the GT1 was the most cumbersome of all the versions. This version required that submission of a job to the Grid network, needed prior knowledge of the machines to execute the job and the client had to do some coding to submit the job. As explained above, the GIS (Grid Information Service) authenticates the user when the user issues the command **Gridproxyinit**. The command **Globusrun** "*Date*" *maxima* for

example, tells the GRIS service to look for the machine called maxima and run the Date service on it. The user interface was through the command line.

The GT2 is also represented in figure 2.3. The user interface is the same as the previous version. In this version the GRIS had a metadata of the machines in the Grid network. It was a better version in that the client did not need a prior knowledge of the machines at the other end. The client could issue a command for example:

Globusrun 'Date' linux, 1mb, time:2 hours. The GRIS takes the command and looks for the appropriate machine to execute the job.

The GT3 gives a windows user interface. Users are presented with a portal that allows them to access Grid services. As shown in figure 2.4, the user runs the command **myproxyinit** on the command line to the MYPROXY server and the server returns a username and password to the user, which can be valid for a month or more. The user can then access Grid resources with this information from any location via a web page. The example in Figure 2.4 shows that a user can open an application, put the necessary input and run it. The user is authenticated with the username and password already created by the MYPROXY server. The Web server verifies this information using the user proxy stored in the MYPROXY server. The Server side is slightly modified to include the Web server (For the Client portal) and a Broker (Handles Service level agreement and allocation of resources.) and the MYPROXY server (issues a username and password and stores user proxy for verification when user accesses the Grid Service).

The GT4 was a merger of Grid Services and Web services thereby making web services technology the standard for communication for Distributed resources and Applications. GT4 is a web-based and significant improvement over previous Globus releases in terms of robustness, performance, usability, documentation, standards compliance and functionality. It makes extensive use of web services to define its interface and structure its components (Foster, 2006). GT4 is used for building grids with services written in a combination of C and java.

GT5 the latest version of the Globus Toolkit, provides a variety of components and capabilities including the following (Vachhani et al, 2012):

- A set of service implementations focused on infrastructure management.
- A powerful standards-based security infrastructure.
- Tools for building new Web services, in Java, C and Python.
- Both client APIs (in different languages) and command line programs for accessing these various services and capabilities.

The GT5 consists of four main components namely:

- Data Management(GridFTP)
- Job Management(GRAM5)
- Security(GSI C, MyProxy, GSI-OpenSSH, SimpleCA)
- Common Runtime(XIO, C Common Libraires)

These components are a huge improvement from all the previous versions aimed at providing a robust toolkit for developing Grid Applications and Middleware.

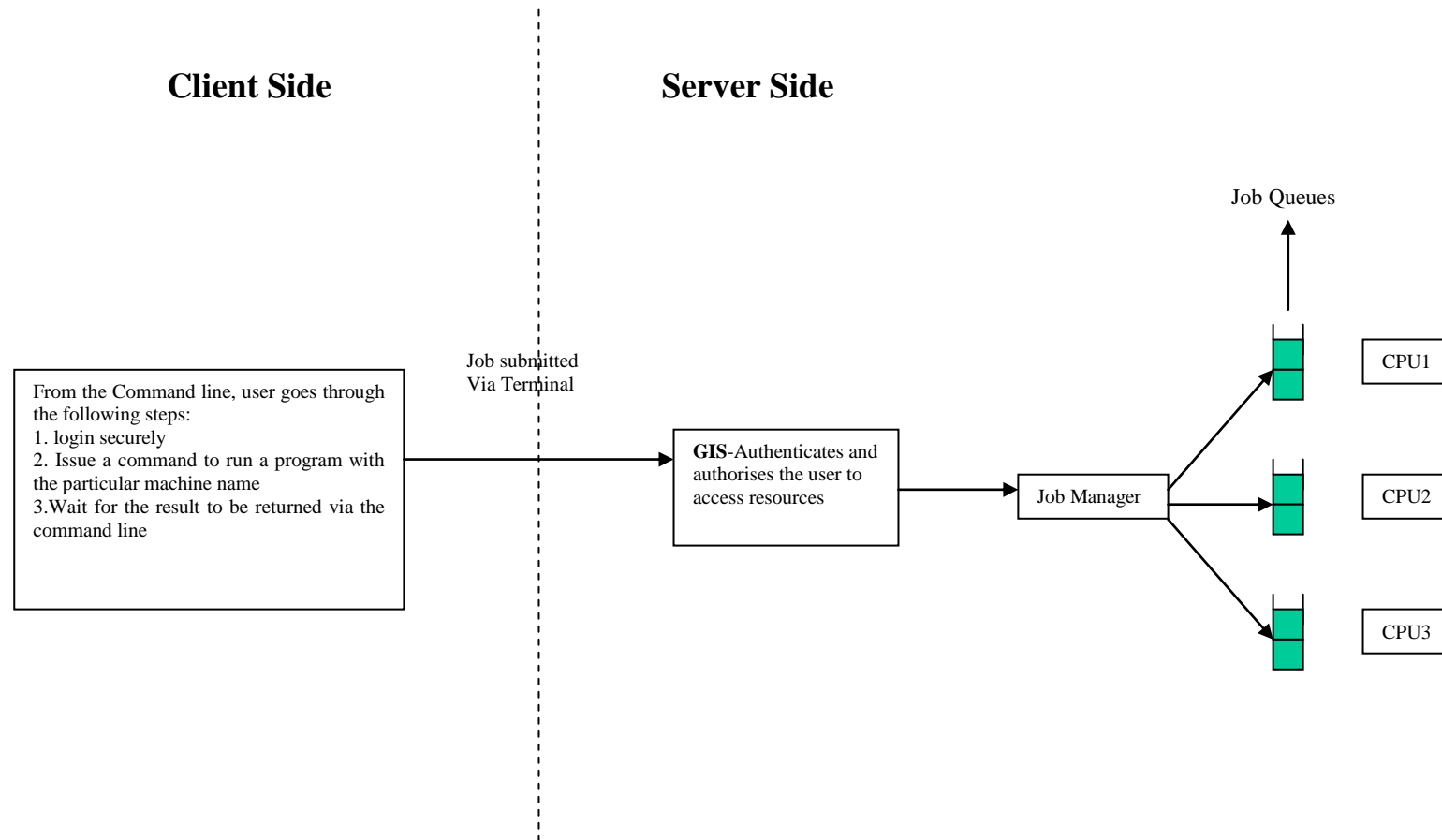


Fig 2.2: The GT1, Shows a brief overview of how a job is submitted to a Grid system using the Globus Toolkit Version 1

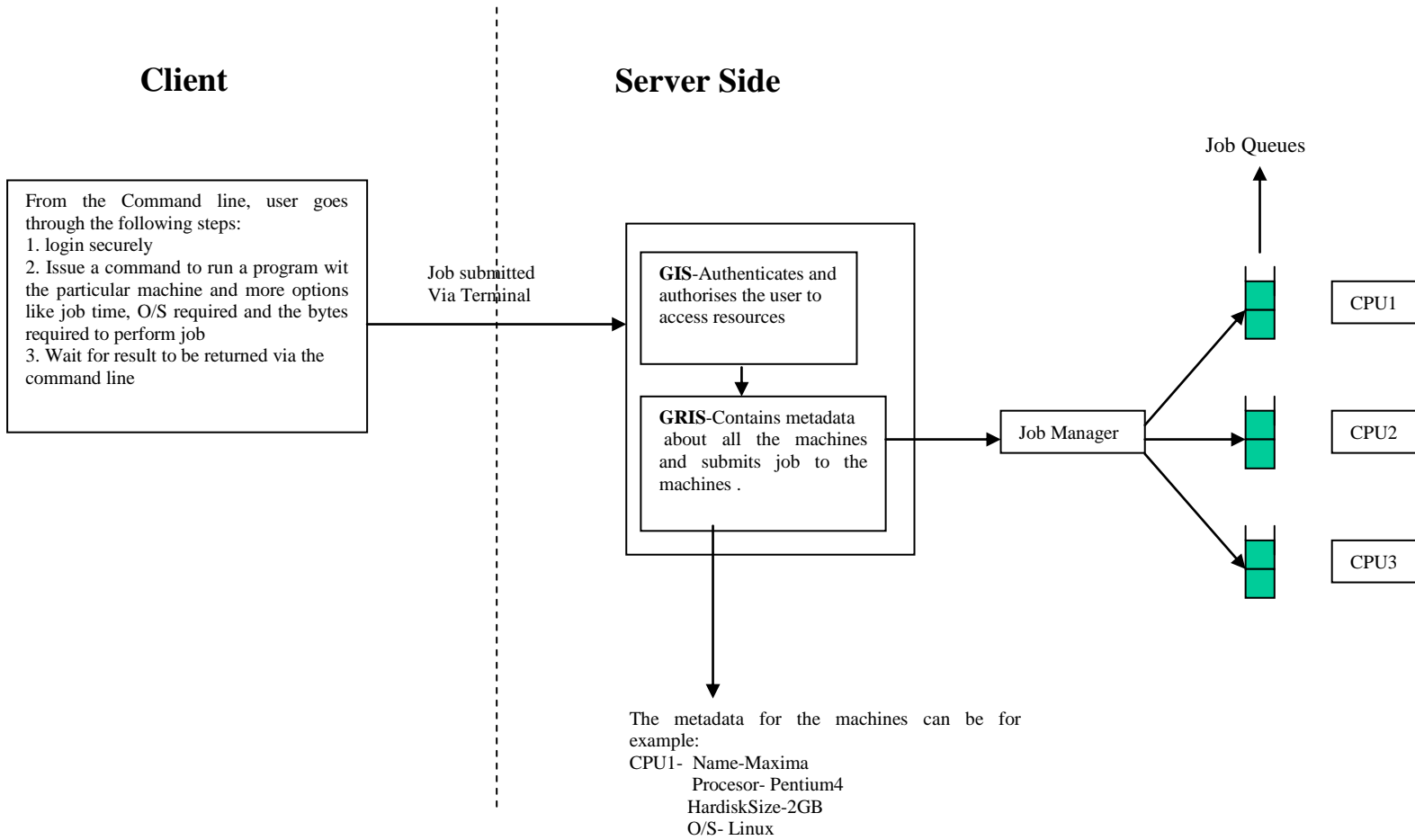


Fig 2.3: The GT2, Shows a brief overview of how a job is submitted to a Grid system using the Globus Toolkit Version 2

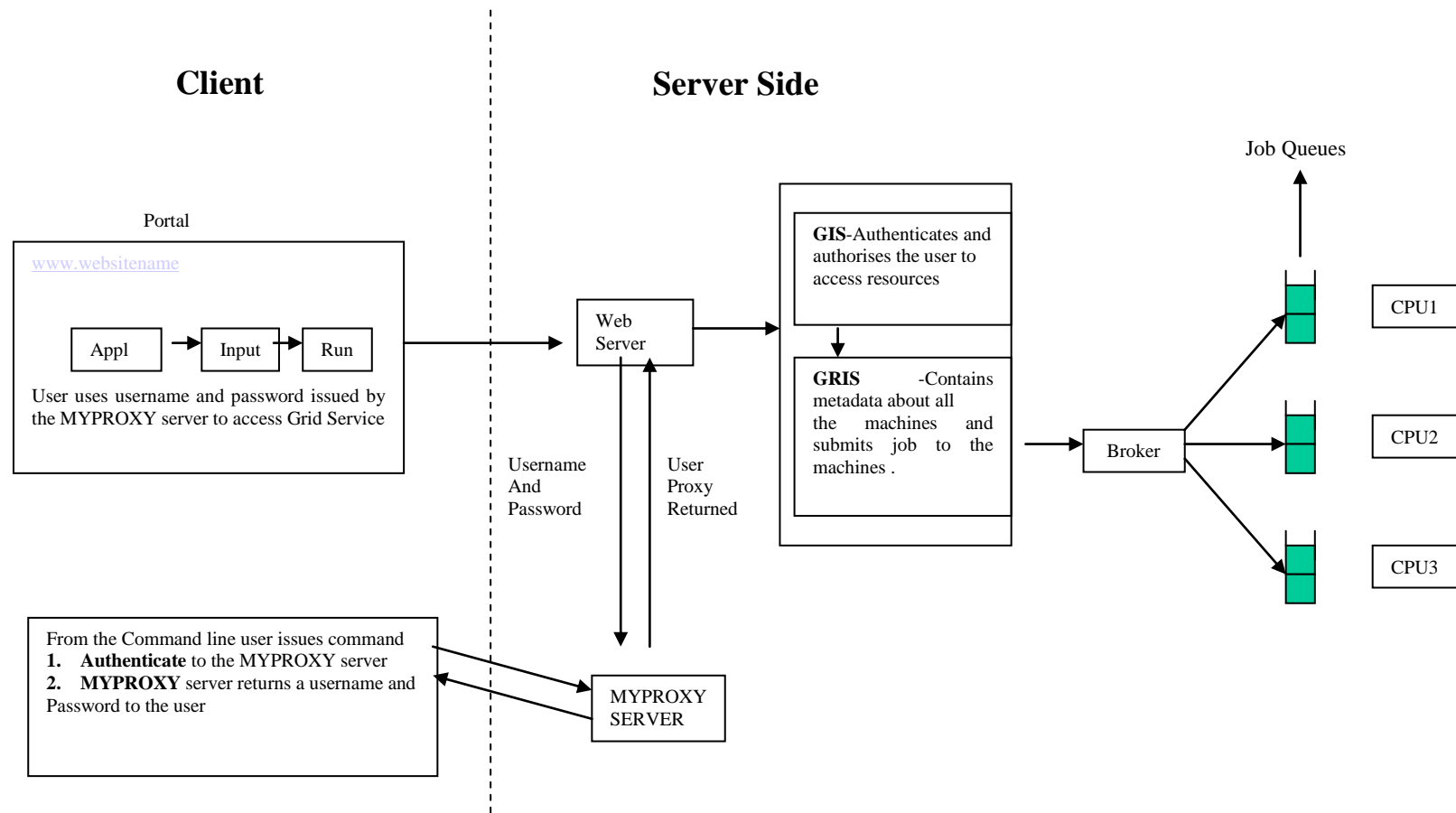


Fig 2.4: The GT3, Shows a brief overview of how a job is submitted to a Grid system using the Globus Toolkit Version 3

2.1.4 Methods for Building Grid Applications

There are two main approaches to building Grid Applications. These are Application based and Portal Based. Each of these has its advantages and disadvantages and the choice of approach has been based on the target user community and their requirements. It is vital that these classifications are made to highlight the features of each approach and why a particular approach was chosen (see Table 2.1).

The Application based approach requires that a piece of software be installed on the Desktop of the user. This software would allow the user access the Grid within the environment that they are already familiar with. The challenge in this approach would be to make sure that the software is easy to install and the user interface hides the complexity of the Grid. The advantage is that the user gets to interact with the desktop and still have access to the application that he or she is used to.

The Portal based approach may be easier to implement since the interface shown to the user is built around the Server side which makes it easier for the developer. The advantage of this approach is that the service can be accessed irrespective of the user location. It still has a drawback of not allowing the user access the application that he or she is already used to on their computer. In most cases the user has to learn an entirely different way of working with their codes.

It is clear from the two approaches that there are users whose daily operations require either of these approaches. The stakeholders here are experimental scientist that would like to make use of the Grid while still close to their everyday programming environment. It must be noted that although they may be programmers in their field they are not interested in programming the Grid but want to use the Grid to improve the execution of their applications(Chin J et al 2004).. Table 2.1 which is a comparison of the two major ways of programming the Grid is also used as a framework for the review of the Grid workflow projects.

Features	Application based	Portal Based
Desktop interaction	Allows user to be in his domain and still benefit from the Grid.	User is presented with an environment that he must comply with. May not allow interaction with the desktop.
Maintenance cost	The maintenance cost is harder as user requirements changes or need for reviews would involve update of the software.	Easier to maintain based on the potential offered by the WWW and the technologies needed to implement it.
Billing	Easier to know which user has used a resource as application is installed on local machine and keeping audit of resource is straight forward.	Keeping audit of resource usage would require that the application is able to handle this statistics, so that the information is reported correctly and accurately.
Security	User would feel more Secure in this environment especially with their data.	Users may have concern about whether their activities are secure.
Scalability	Require extra coding for a new requirement.	Easier to modify for small or large applications.
Ease of use	Desktop software that aims to extend the functionality of how the user currently runs his compiled programs.	User has to learn a new ways of executing their codes
Installation	Requires that a piece of software be installed on desktop.	No installation required.
User Community	This approach is usually general for use with any body that has a code to run on the Grid	This approach usually has the tendency of being implemented for particular community of users making hard for other user community to benefit from it.

Table 2.1: A comparison of the two approaches to implementing a Grid interface.

2.1.5 Grid WorkFlow Projects

This review shows how earlier Grid projects were built with a lot of emphasis on functionality rather than usability and how this can be used by intended users are discussed below. This is analysed with respect to how Grid application developers built applications, deployed it and how scientific users can benefit from that deployment. This review has also helped in gathering user requirements for the final interface built.

2.1.5.1 MyGrid Project

^{my}Grid project is an e-Science project with an emphasis on Information Grid. The target users for this project are a community of biologists who would like to share

computational resources, large-scale data movement and replication for simulations. The ^{my}Grid project built services for data and application integration such as resource discovery, workflow enactment and distributed query processing. The project was first built using the XML-based web services but migrated to the ‘Open Grid Services Architecture’ (OGSA) (Stevens et al 2003). The ^{my}Grid project has been able to deal with some of the problems encountered with other workflow projects by focusing on the services that allows for handling failures, recovery etc.

The ^{my}Grid services were deployed using the portal approach and methods were exposed as web services to be used by the users. The three main services include (Robert D.Stevens et al, 2003):

- Services for forming Experiments.
- Services for discovery and metadata management.
- Services for supporting e-Science.

Users access these services via a web interface. The services for forming experiments allow users to distribute and retrieve data in different bioinformatics formats. Users can then construct workflows using the workflow engine to call bioinformatics services. Users are able to discover services using metadata information. The services for supporting e-Science provided users with notification to when a workflow may be re-run or when new or updated data and analytical software became available; users can register to receive updates and the services register the kind of updates they provide. The personalisation service allowed users to have different views of the ^{my}Grid information repository (mIR) and the provenance service can provide information for auditing and to enable the use of notification events generated by services to determine if a workflow needs to be re-run

2.1.5.2 Triana

The Triana project was aimed at building a problem-solving environment originally for analysing waves signals. It is a graphical programming environment that allows users to compose and execute distributed workflows. Triana services were deployed using the application based approach but can also be used as a Web services composition toolkit.

The main objectives of the Triana project (Majithia et al, 2003) are:

- Hide the low-level details of composing Web services.
- Allow the user to focus on design of workflow at the conceptual level.
- Specifically allow the user to graphically and transparently :
 - Discover relevant services.
 - Compose services.
 - Invoke composed graph.
 - Publish Composite services.

Triana offer to users three services namely: Discovery services, composing services and publish services. User's actually install the Triana application tool on their desktop and then open it to use any of these services.

Figure 2.4 shows a snapshot of how a workflow is constructed in Triana. Users are able to discover services they need by using the UDDI or specifying a WSDL location or matching a search criteria. When these services are discovered users can then compose a workflow by dragging and dropping the services onto the workspace as shown above. The services are connected with pipes and local tools can also connect with web services. These graphs are generated in BPEL4WS (Business Process Execution Language for Work Flows). The user can then run the workflow and get results. Users are also able to compose services and publish for discovery.

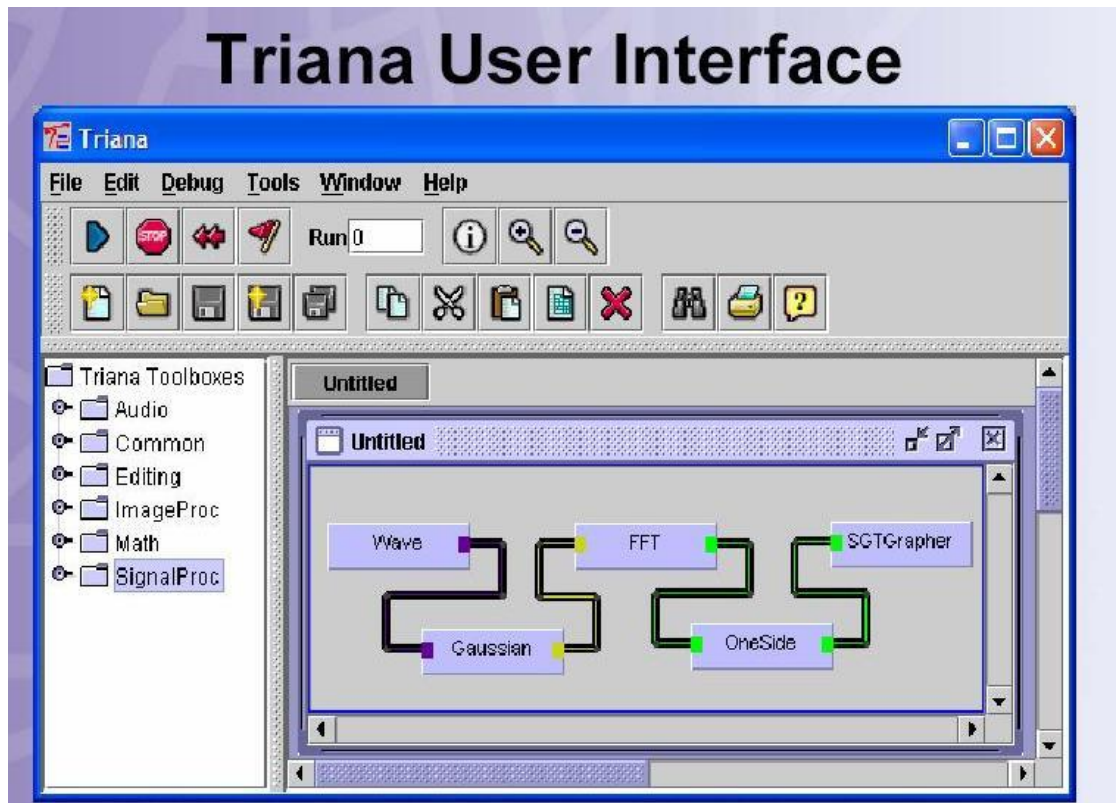


Fig 2.5: A snapshot of Triana User Interface

2.1.5.3 Geodise

Grid enabled optimisation and design search for engineering (GEODISE) is another ESPRC funded project involved with the use of distributed computing resources as applied to design optimisation in the field of computational fluid dynamics (CFD). The Project aimed to aid engineers in the design process by making available a suite of design search and optimisation tools and CFD analysis package integrated with distributed Grid-enabled computing, databases and knowledge management technologies (Xu, Cox 2003).

GEODISE is deployed as an application that presents to design engineers Grid services within the Matlab environment as Matlab function that conform to Matlab syntax. The resources needed by the engineers are exposed via a web-based portal. Users of this tool are engineers used to the Matlab working environment who work with a Workflow tool (a standalone GUI application) implemented in java and runs on any platform. Users are presented with a component view displaying components defined as Matlab functions, workflow views allowing users to create

workflow or reuse an existing one stored on a local database and compute view which shows the distributed servers for a user to choose for their job execution.

2.1.5.4 DAME

The Distributed Aircraft Maintenance Environment (DAME) is an e-Science pilot project demonstrating the use of the Grid to implement the design and development of decision support systems for diagnosis and maintenance, in which geographically distributed resources, actors and data are combined within a virtual organisation [60]. The project is borne out of a need for aircraft companies to study and analyse the volumes of engine data that is generated each time that a plane has landed. These data could be in terabytes. There was a need to find a system that can handle such volume of data and one that can collate this data simultaneously from different locations. There was also a need to implement a Grid enabled environment into such a system.

DAME consisted of different services that were integrated together. The engine data service is a replicated service that handles communication between the ground station and Grid data repositories. The Data storage and mining service consists of the AURA pattern matching and engine system used to search raw and archived engine data. The Engine modelling service takes parameters from flight data and runs models of the engine to infer the current state of the engine. The case-based reasoning service uses case-based reasoning (CBR) to improve the knowledge base and capture DP methods in a procedural way. The maintenance interface service organises all the interactions with stakeholders involved in taking remedial actions in response to a diagnosis or prognosis.

Users basically use a Signal Data Explorer (SDE) tool as a front end for exploring and searching time series data accessible via the SRB (Storage Resource Broker). The SDE connected to a single Pattern Match Controller (PMC) Grid service then requests for a distributed data to be searched and the PMC will in turn connect with the other sites to search the data. The PMC is deployed as a Grid service at each data centre coupled with a local search engine, which worked on data stored in each local repository. The results are then displayed for the user through the SRB. All data access was via the SRB and results were also obtained through the SRB.

2.1.5.5 Summary of Grid Workflow Projects reviewed

The Grid workflow projects were reviewed and analysed with three criteria namely; how the scientific application developers built Grid application, how they were deployed and the usability of the interface.

The myGrid project does not allow integration of user's local application with the application exposed as services on the web. Users would have to describe their application as services to allow them talk with other services. Not all users' application code can be described easily as workflows. It is geared towards a particular user community and would not work for any user that just wants to run their code on the Grid.

Triana allow users to compose local tools with services on the web. These local tools are actually described as some form of workflow language that the web services can relate too. Not all users' application code can be described easily as workflows. Users could not just have an executable and expect that it could be connected with the existing services in a straightforward manner. With Triana, most of the codes already exist and are for signal analysis and my interaction with it did not easily accommodate an entirely new compiled code.

The Geodise project also required that users must describe their tools as Matlab functions and users must create workflows to submit jobs for execution. Not all users' application code can be described easily as workflows in Matlab. Again users have to learn new tools just to be able to use the Grid. The Dame project also expects the user to have knowledge of workflows. The interface to use this tool is quite complicated and geared towards expert users of the Grid and would throw a novice off balance.

Review of the four Grid projects brought about a common problem with the interfaces. The Usability aspects of the projects were not properly looked at as to how users would effectively interact with the system. Also there was always a need for users to learn new tools to successfully use the Grid services being offered. These projects have been able to demonstrate the great functionality offered in Grid

computing but the actual usage of these interfaces still remained a big challenge (Chin J et al 2004).

2.2 Usability

System functionality addresses whether the system is capable of satisfying the needs and requirements of the user; usability addresses how well users can use the functionality provided by the system (Nielsen, 1993). A number of definitions of usability have been provided by modern literatures. The International Organisation for Standardization (ISO) defines usability as the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment (Faulkner, 2000). Focusing on both ease of learning and ease of use, (Shackel 1991) propose that usability is concerned with the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfil the specified range of tasks, within the specified range of environmental scenarios. (Dumas and Redish 1993) believe for a system to be usable, its users must be able to use the product quickly and easily to accomplish their own tasks. These four points capture the essence of usability:

- Usability focuses on users,
- People use products to be productive,
- Users are busy people trying to accomplish tasks,
- Users decide when a product is easy to use.

2.2.1 Usability principles and guidelines

When designing a system that involves a current or a new user interface, it is important to follow these important design principles (Dix et al, 2003):

1. Focus early on users and tasks which involve determining how many users are needed to perform the tasks and the appropriate users for the domain. Define the tasks and how often these tasks are performed
2. Empirical measurement, which involves testing the interface early on with users who come in contact with the interface frequently. Establish quantitative usability specifics like: the number of users performing the task(s). the time to complete the tasks(s) and the number of errors made during the task(s)

3. iterative design: When users, task and empirical measurements are determined, perform the iterative design steps
 - i. Design the user interface
 - ii. Test
 - iii. Analyze results
 - iv. Repeat steps (i-iii)

2.2.2 The Classic Software life Cycle: The Waterfall Model

The waterfall model is a well-defined development approach as stated by (Avgeron, Cornford 1998). The waterfall model is an interactive design software development process, viewed as any other engineering process. The advantages of using the waterfall model are that it helps to organize and conceptualize the various tasks involved in information systems development (Matravers 2001). This provides a framework to structure the project. The waterfall model is concerned with developing a system from initial conceptualization unlike other development models that require an existing system. According to (Sommerville, 2001), the principal stages of the model map onto the following fundamental development activities:

1. Requirement analysis and definition: The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification
2. System and Software design: The systems design process partitions the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental system abstractions and their relationships.
3. Implementation and unit testing: During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.
4. Integration and system testing: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5. Operation and Maintenance: Normally (although not necessarily) this is longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier

stages of the life cycle, improving the implementation of system units and enhancing the systems services as new requirements are discovered.

2.2.3 Iterative Design and Prototyping

According to (Dix et. al 2003) the requirements for an interactive system cannot be completely specified from the beginning of the life cycle. The only way to be sure about the features of the potential design is to build them and test them out on real users. The design can then be modified to correct any false assumptions that were revealed in the testing. Therefore iterative design tries to solve the problem of incomplete requirements specification by cycling through several designs incrementally improving upon the final product with each pass. (Dix et al 2003) advises to use iterative design in conjunction with other more principled approach to interactive system design.

Iterative design is described by the use of prototypes artefacts that simulate some but not all features of the intended system. There are three main approaches to prototyping:

Throwaway: The prototype is built and tested. The design knowledge gained from the exercise is used to build the final product but the actual prototype is discarded.

Incremental: The final product is built as separate components one at a time. There is one overall design for the final system but it is partitioned into independent and smaller components. The final product is released as a series of products with each subsequent release including one more components.

Evolutionary: The prototype is not discarded but serve as a basis of design for the next iteration of design. The actual system is seen as evolving from a limited initial version to its final release.

Techniques for prototyping

There are many techniques available that allow you to prototype what an intended system proposes to achieve (Dix et al 2003):

Storyboard

This is a graphical depiction of the outward appearance of the intended system without any accompanying system functionality. They can be mocked up without the aid of any computing resource

Limited functionality simulations

Here more functionality is built into the prototype to demonstrate the work that the application will accomplish. Some portion of the functionality of the system is simulated by the prototype. The designer can rapidly build graphical and textual interaction objects and attach some behaviour to those objects which mimics the system functionality. Once the simulation is built, it can be evaluated and changed rapidly to reflect the results of the evaluation study with various users. HyperCard a well known prototyping tool can be used for this purpose.

High-Level programming support

This can be provided by a user interface management system (UIMS). The conceptual model put forth for interactive system design is to separate the application functionality from the presentation. It is then possible to program the underlying functionality of the system and to program the behaviour of the user interface separately. The job of the UIMS is to allow the programmer to connect the behaviour at the interface with the underlying functionality.

2.2.4 User-Centred Design

User-centred design (UCD) is an approach to software development that focuses specifically on making products usable. The goals of UCD [72] is to obtain a software that is

1. Easier to understand.
2. Improve the quality of users by reducing stress and improving satisfaction.
3. Significantly improve the productivity and operational efficiency of individual users and consequently in their work place.

According to the International Standards Organisation (ISO) [72], the ISO 13407 standard mandates that there are four essential user-centred design activities that should be undertaken to incorporate usability requirements into the software development process:

1. **Understand and specify the context of use:** Here you collect relevant contextual information like the characteristics of the intended users, the tasks the users will perform and the environment in which the users will use the system.
2. **Specify the user and organisational requirements:** This involves formulating an explicit statement of the user-centred requirements for the new software based on the context of user description obtained earlier. Important elements includes the identification of the range of users, provision of a clear statement of design goals, an indication of the appropriate priorities for the different requirements and evidence of acceptance of the requirements by the stakeholders.
3. **Produce design and prototypes:** This can be used to foster greater communication between the users and the designer of the software. Design solutions are explored by creating simple mock-ups of the proposed system and then presenting them to a representative sample of users. The initial design can then be refined after getting the user feedback.
4. **Carry out user-based assessment of the system or prototype:** This is usually done with the following steps:
 - Device an evaluation plan.
 - Perform data collection and analysis.
 - Report the results and recommendations for change.
 - Iterate the above activities until the design and usability objectives are met.
 - Track changes maintenance and follow-up.

2.2.4.1 Usability Engineering

(Bennett et. al 1988) and (Nielsen J 1992) suggested that usability engineering goals used in the design process is another approach to user-centred design. The emphasis for usability engineering is in knowing exactly what criteria will be used to judge a product for its usability.

This approach test products usability based on measurements of users experience with it. The danger here is that much of the work accomplished in interaction involves the functional architecture of the system and cognitive capacity of the users. These two components have to be observed in order to arrive at a meaningful measure. But it is very hard to derive measurements of activities beyond the interaction of the physical interface of the system so this approach is limited in its application

In relation to software life cycle usability engineering includes a usability specification as part of the requirements specification that concentrates on features of the user-system interaction which contribute to the usability of the system. Various attributes of the system are suggested as gauges for testing the usability. For each attribute, six items are defined to form the usability specification of that attribute.

2.2.5 Design Rationale

This is an area of usability that does not really fit into the interactive design process. Rather it is the information that explains why a computer system is the way it is including its structural description and its functional or behavioural description. There are many reasons why a design rationale is necessary:

1. It can help to understand what critical decisions were made during the design, what alternatives were investigated and why a particular alternative was chosen.
2. Accumulated knowledge can be reused to transfer work that was done in one situation to another situation which has similar needs.
3. The design rationale forces the designer to deliberate more carefully about design decisions.
4. There is usually no single best alternative. The designer is usually faced with a set of trade-offs between alternatives. For example a graphical interface may involve a set of actions that the user can invoke by the use of mouse. The designer must decide whether to present action as a 'button' on a screen which is always visible or hide all of the actions in a menu which must be explicitly invoked before an action can take place. The button option maximizes the operation visibility but the menu option takes

up less space. The designer then need to determine which criterion for evaluating the option is more important.

5. The usability of an interactive system is very dependent on the context of its use. Capturing the context in which a design decision is made will help later when new products are designed.
6. It is very important for the designer to indicate all the alternatives that have been investigated. Later on it can be determined if he/she considered the best solution or thought about it and discarded it. This kind of accountability for design is very good.

2.3 Evaluation criteria Technique

Evaluation in usable interactive systems entails assessing our system designs and testing to ensure we have built a system that behaves as we expect and meet the user requirements. Ideally, evaluation should occur throughout the software lifecycle as it can be useful in providing constructive feedback that can be useful in modifying the design to better suite the user's requirements. The main objective of evaluation is to: assess the extent and accessibility of the systems functionality, to assess the user's experience of the interaction and to identify any specific problems with the system (Dix et al 2003). Evaluation techniques are described under two heading: expert analysis and user participation.

2.3.1 Evaluation through expert analysis

This is an evaluation technique that is useful from the design of the system up to the final system implementation and is usually tested by an evaluator rather than the actual user of the system. This technique can be used to evaluate even the design of the intended system. It is therefore difficult to get the exact interaction assessment from a system that is just in the prototype stage. The basic intention is to identify any areas that are likely to cause difficulties because they violate known cognitive principles or ignore accepted empirical results (Dix et al 2003). Common evaluation techniques through expert analysis include cognitive walkthrough, heuristic evaluation, use of models and use of previous work.

Cognitive walkthrough

The origin of this evaluation technique is the code walkthrough familiar with software engineering. Walkthroughs requires a detailed review of a sequence of actions and in code walkthrough this sequence represents a segment of the program code stepped through to check that certain characteristics are met (Dix et al 2003). In cognitive walkthrough this sequence of actions refers to the steps that an interface will require for a user to perform, in order to accomplish some known tasks. The evaluator then ‘step through’ that action sequence to check it for potential usability problems (Dix et al 2003). Cognitive walkthroughs involve simulating a user’s problem-solving process at each step in the human-computer dialog, checking to see if the user’s goals and memory for actions can be assumed to lead to the next correct action (Nielsen and Mack 1994). The main focus of this technique is to determine how easy the system is to learn through exploration. For this technique, you need the following four things:

- A specific prototype of the system whether complete or not.
- A description of the task the user is to perform on the system.
- A written list of actions needed to complete the task with the proposed system.
- An indication of who the users are and what kind of experience and knowledge the evaluators can assume about them (Dix et al, 2003).

The evaluators, given this information above, then step through the action sequence of the particular task and try to critique the system and tell a story about its usability. For each action, the evaluators try to answer the following four questions:

- Is the effect of the action the same as the user’s goal at that point?
- Will users see that the action is available?
- Once users have found the correct action, will they know it is the one they need?
- After the action is taken, will users understand the feedback they get?

Heuristic Evaluation

A heuristic is a guideline or general principle or rule of thumb that can guide a design decision or can be used to critique a decision that has already been made. This technique can be performed on a design specification and can be used on prototypes, storyboards and fully functional systems. The ten heuristics by Jakob Nielsen can be found at [79]. The general idea is that several evaluators independently critique a

system to come up with potential usability problems. It is important that these evaluations be done independently and Nielsen's experience indicates that between three and five evaluators is sufficient which results to 75% of the overall usability problems being discovered. To aid the evaluators the ten guidelines are used and in some cases heuristics that are particular to that system can supplement any of the ten. The evaluator then assesses the system and note which of the heuristics is violated based on an overall severity rating on a scale of 0-4:

- 0=I don't agree that this is a usability problem at all.
- 1=Cosmetic problem only: need not be fixed unless extra time is available on the project.
- 2=Minor usability problems: fixing this should be given low priority.
- 3=Major usability problems: important to fix, so should be given high priority.
- 4=Usability catastrophe: imperative to fix this before product can be released.(Nielsen, 1993).

Once each evaluator has completed their separate assessment, all of the problems are collated and the mean severity ratings calculated.

(Nielsen, 1993) ten heuristics are:

1. **Visibility of system status:** Always keep users informed about what is going on in the system through appropriate feedback at reasonable time.
2. **Match between system and real world:** The system should speak the user's language with words phrases and concepts familiar to the user.
3. **User Control and freedom:** Users should be able to undo or redo an action that was taken by mistake.
4. **Consistency and standards:** Users should not have to wonder whether words, situations and actions mean the same thing in different contexts.
5. **Error prevention:** Make it difficult to make errors.
6. **Recognition rather than recall:** Make objects, actions and options visible.
7. **Flexibility and efficiency of use:** Allow user interaction to be such that it can tailor for both experience and inexperienced users.
8. **Aesthetic and minimalist design:** Dialogs should not contain information that is irrelevant or rarely needed.

9. **Help users recognise, diagnose and recover from errors:** Error messages should be explained in plain language not codes.
10. **Help and documentation:** it may be necessary to provide this to properly guide the user.

Model-based evaluation

This involves the use of models. Certain cognitive and design models provide a means of combining design specification and evaluation into the same framework. For example the GOMS (goals, operators, methods and selection) model predicts user performance with a particular interface and can be used to filter particular design options (Dix et al, 2003). Dialog models can be used to evaluate dialog sequence for problems such as unreachable states, circular dialogs and complexity. Models such as state transition networks are useful for evaluating dialog designs prior to implementation.

Group-based expert walkthrough

This particular usability inspection method is fairly new in usability methods and involves engaging work-domain experts as the evaluators, as against the three expert based method described above which involved usability experts. This method was particularly developed to support non-usability experts as evaluators (Sjoberg et al 2010). The work-domain experts are: (a) potential end users with direct experience from the work-domain or (b) persons with extensive secondary knowledge of the work domain (Folstad, 2007). This particular technique argues that usability experts usually have little knowledge of the interactive systems that they are usually told to inspect and as such it would be worth exploring using work-domain experts as evaluators of these systems. Before evaluation, usually a usability expert would take the work-domain experts through the standard usability basics so that these experts can correctly report usability problems that exist in the particular system to which they are analysing.

2.3.2 Evaluation through user participation

This technique involves the actual users testing the system. User participation in evaluation tends to occur in the later stages of development where there is at least a working prototype of the system in place. This could range from a simulation of the

systems interactive capabilities without its underlying functionality through a basic functional prototype to a fully implemented system (Dix et. al. 2003). The techniques under these categories can still help in positive contribution in the early design stages (user's requirement capture). The techniques in this category are described below:

Empirical methods: experimental evaluation

This involves the use of a controlled experiment which provides empirical evidence to support a particular claim or hypothesis. The evaluator chooses a hypothesis to test, which can be determined by measuring some attribute of participant behaviour. The participants are chosen to match the expected user population as closely as possible. In this technique it is preferable to use the actual intended users of the domain but if that is not possible then a group of users that have similar background can be used. They should have similar computer background and a fair knowledge of the task of the domain. If the intention is to run a controlled experiment then the amount of users should be at least 8 as recommended by (Dix et. al. 2003) because statistical analysis would have to be performed on the results from the testing.

Observational techniques

This involves observing users as they interact with the system. They are given a set of predetermined tasks and the evaluator then records the user's actions using a variety of techniques namely:

Think aloud and cooperative evaluation.

Think aloud and cooperative evaluation is a form of evaluation where the user is asked to talk through what he is doing as he is being observed. The usefulness of this technique is largely dependent on the effectiveness of the recording method and subsequent analysis. The record of an evaluation session of this type is known as a protocol and the different methods include: Paper and pencil, Audio recording, Video recording, Computer logging and user notebooks. Further reading can be gotten from (Dix et al 2003).

Query techniques

This type of technique involves asking the user about the interface to get an idea of the user's view of the system. It helps to collect information about the user's requirements and task and helps to get the users viewpoint directly and may reveal issues that may not have been considered by the designer. These techniques are usually simple and easy to administer but it may also be difficult to get accurate feedback about alternative designs if the user has not experienced them, which limits the scope of the information that can be gathered (Dix et. al., 2003). The two main types of query techniques include *interviews* and *questionnaires*.

Interviews provide a direct and structured way of gathering information. It has the advantage that the level of questioning can be varied to suit the context and that the evaluator can probe the user more deeply on issues about the interface that may prove interesting to the evaluator. Interviews can be effective for high-level evaluation, particularly in eliciting information about user preferences, impressions and attitudes and can thus reveal problems that have not been anticipated by the designer or that have not occurred under observation (Dix et al, 2003).

Questionnaires is another querying method that is less flexible than interviews as the questions asked are already fixed in advance and could be less probing than intended. The main advantage is that it can be used to reach a wider range of users, requires less time to prepare and can be analysed easily. It could also be used in various stages of the software life cycle from requirement capture to task analysis to evaluation to better understand the user's needs and preferences. It is recommended that the evaluator determines the purpose of the questionnaire: What information is sought? It is useful to determine how it will be analysed (For example do you want specific, measurable feedback on particular interface features, or do you want the users impression of using the interface?)(Dix et al, 2003). There are a number of styles of questions that can be included in the questionnaire. These include general, open-ended, scalar, multi-choice and ranked.

2.3.3 Establishing evaluation criteria

Identifying the issues involved in a Usability Evaluation is not adequate in order to perform evaluation. It is also required to establish a series of criteria to measure

Usability. These criteria should be both measurable and fulfil user's needs. A survey of the literature (Rengger 1991) has identified four classes of performance measures; Goal achievement (accuracy and achievement), work rate (productivity and efficiency), knowledge acquisition (learnability and learning rate) and operability (error rate and function usage, flexibility). These measures can be achieved with whatever evaluation technique that is used for the evaluation of the interface.

2.4 Previous Research on Usability of the Grid.

2.4.1 Grid Middleware aimed at Grid Usability

Since the advent of the Grid, a lot of middleware like Globus have been initiated by different Grid projects in the effort to make programming the Grid easier for Grid application developers. Some of these middleware's attempt at this is discussed in the next section.

2.4.1.1 The GridLab Project

The GridLab project aimed to provide new capabilities for applications to exploit the power of Grid-computing bridging the gap between application needs and existing Grid middleware (Allen Gabrielle et al 2003). The aim of GridLab was to provide users and application developers with a simple and robust environment enabling them produce applications that can exploit the full power and possibilities of the Grid. The GridLab architecture comprised of the UserSpace (User Application and Gat) and the Capability Space (GridLab Services and Third Party Services: GIS or GRAM, system services, libraries).

The GridLab project was demonstrated with exemplary GridLab components: The GAT (the application interface to Grid environments), Triana (One of GridLab applications) and the GridLab Scheduling Service (a GridLab service). The main goal of the GridLab project was to provide a software environment for Grid-Enabling scientific applications by providing API through which the application can access and use available resources. This API is concentrated in the GAT. The GAT was designed to be able to interact with all types of capability providers (An entity providing a specific capability eg OGSA, RPC, CORBA).

GAT aimed to provide the functionality through a carefully constructed set of generic high-level APIs through which an application will be able to call the underlying Grid services. An example is when an astrophysicist accesses the GEO600 portal to perform any of his jobs. The underlying brokering tool will try to find fast affordable machines to perform his job and by clicking to accept the portals choice, initiates a complex process by which executable and data files are automatically moved to these machines by the scheduling and data management tool and then the analysis is performed.

How GAT Works

The user application invokes the Gat via some GAT-API call and the GAT engine then queries the capability registry for a suitable adaptor for that API call and the selected adaptor is invoked. On failure, the Engine will select the most suitable adaptor from the registry. An adaptor can interface to one or more capability provider in order to provide a particular functionality. The API definition is driven by the application developer and user groups and not by the underlying Grid environment and what it offers. So the application programmer then has to incorporate Gat functions in his own code to access some specific Grid services. Gat initialization involves all the adaptors (underlying Grid resources) registering the capabilities they provide with the GAT Capability Registry. Some Gat functions, which lead to a successful use of Grid resources, can be achieved with the three Gat functions in the order presented below (Allen Gabrielle et al 2003):

1. **context = GAT_Init (Requirements):** In this instance context denotes an opaque object holding persistent GAT information (like security credential and the internal state of the GAT Engine). Requirements specifies any specific information needed for the initialisation like the initial adaptors.
2. **GAT_FindResource (context, Requirements, Resource)** queries the environment for a resource matching a number of requirements with the Requirements parameter having the resource requirements of the application and the Resource parameter holding information about a matching resource which is returned by the GAT-API.
3. **GAT_SubmitJob (GAT_State, Resource, Job)** can use the information from the previous GAT call to, for example start a job on the machine. Here Job is an opaque object holding all the information needed to execute the job

The GAT engine will then find different adaptors providing the functionality to run a job on some resource and if for any reason an adaptor fails, the engine will invoke the next adaptor that can handle that job

GridLab Services

The GridLab services are designed to complement and complete the existing Grid infrastructure and also to provide functionality needed by GridLab applications in order to be usefully deployed in such environments. As an example, the GridLab Resource Management Services is described below.

GridLab Resource Management Services: This is system of interoperating services providing resource management, scheduling and job execution capabilities on the Grid. This services aims to answer questions concerning how to map activities such as computation or data transfer onto sets of resources belonging to different organizations in ways that will meet user requirements for performance cost security and other metrics corresponding to quality of service. The goal is to create a super scheduler responsible for making decisions to achieve the best possible resource utilization. This consists of other services described below

- *The GridLab Resource Management System:* This provides scheduling mechanisms that fit the needs of users and application with regard to local policies.
- *GRMS Configuration and Policy Services:* These are responsible for GRMS configuration. This service allows users (GRMS administrators) to enforce different resource management policies which can be mapped onto different groups of users.
- *Core GRMS Services:* This includes Job Receiver, Resource Discovery, Resource Evaluation, Brokering Prediction, QOs Resource Reservation and Resource Estimate Services. These are all described in (Gabrielle et al 2003).
- *Job Execution Service:* these addresses resource scheduling and computing access.
- *Infrastructure service:* these are the services in the GRMS that interact with other services in the underlying Grid. These allow the GRMS to query information from the Grid environment to utilize lower level capabilities for the GRMS services like security. This further has some sub services described in (Gabrielle et al 2003).

All the above mentioned services are presented to any application running on the Grid through the GAT (Grid Application Toolkit).

2.4.1.2 GridGain

GridGain is a java based distributed computing middleware. It includes state-of-the-art support for computational grids, data grids and auto-scaling on any grid or cloud infrastructure [68]. GridGain supports java and scala programming languages. GridGain offers to business applications the following benefits [70]:

- Work in a zero-deployment mode.
- Scale or down based on demand.
- Cache distributed data in data.
- Collate data and computations for best utilization of resources.
- Run sql and text queries against cache data.
- Speed up long running task using MapReduce.
- Use distributed thread pools.
- Evenly distribute the workload on the grid.
- Effectively exchange messages.
- Auto-discover all grid resources.
- Execute closures on the grid
- Grid-enable existing java and scala code

Basically, GridGain allows users parallelize the execution of their piece of code onto a set of computing resources (heterogeneous or homogeneous) which can be a laptop, desktop computer, mainframe installed with java 5 or a higher compatible java. According to (He Yunhui et al, 2010) GridGain in comparison with Globus toolkit has the following two advantages: 1.) It is easier to install and deploy which is helpful so the programmer can focus on the application for the Grid technology and 2.)It can run perfectly on windows platforms.

The current version GridGain 3.0 is the first data grid featuring zero-deployment capability enabling users to simply bring up default GridGain nodes online and they immediately become part of the data grid topology and can store any user objects without the need for explicit deployment of user's classes [70]. This version also

comes with GridGain Visor – a pluggable and scriptable command line management and monitoring tool with the following key features [70]:

- Allows to “script” various operations on GridGain deployment.
- Interactive and command modes.
- Fully extensible via user defined pluggable commands.
- Seamless connectivity to the running GridGain deployment.
- Some available commands to allow you, review and monitor topology, execute Grid tasks, monitor status, get statistics and query data Grid.

2.4.2 User-friendly Grid projects

Current projects aimed at improving the usability of Grid-environments are discussed below. The criteria for analysis are based on the job submission process and the interface for user interaction as these are the main usability requirements that have been identified for this project.

2.4.2.1 The eMineral Project

This project [62] is aimed at building usable grids for computational scientists in several domains within the physical sciences (see figure 2.5). This group of users are comfortable with working with shell tools rather than graphical interfaces because they would like to have a degree of control over their applications. The eMineral project therefore aimed to tackle this problem by developing the ‘my_condor_submit’(MCS) tool which provides a simple scriptable interface to Globus, a flexible interaction with the storage resource broker, metascheduling with load balancing within a grid environment and automatic metadata harvesting(Walker et al 2006).

These set of users have traditional ways with which they interact with the Grid. An average user would take the following steps to interact with the Grid on a day to day basis

1. Prepare input data files, job scripts, and application.
2. Transfer them to the compute resource for running the job.
3. The user logs on to these compute resource.

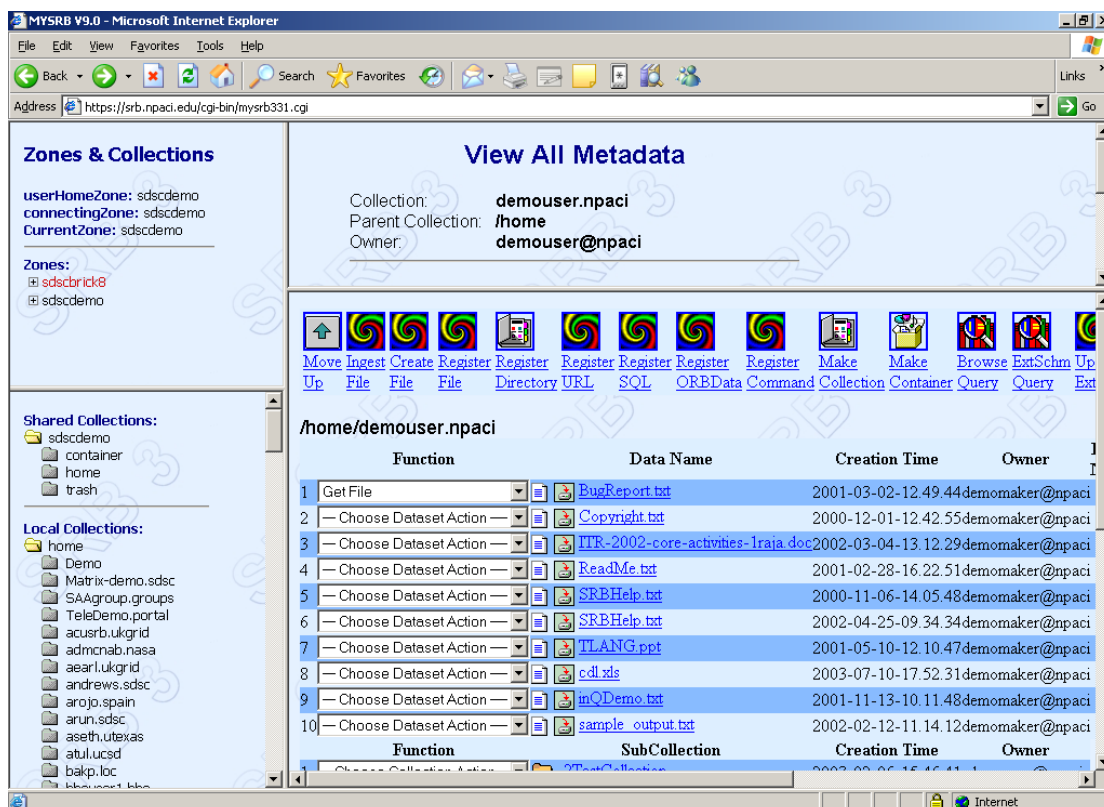


Fig 2.6 Screen shot of the e-Mineral Web-based tool (TobySRB)

4. Compiles the application if an executable does not exist.
5. Job submitted to appropriate queue.
6. Later come to monitor the status of the job.
7. On completion of job, results transferred back to local computer usually user's desktop.

To build usable interfaces the project team have used the bottom-up policy of making sure the scientists work closely with the developers to set the overall user requirements and specification. This approach would allow developers build prototypes, have the scientists use them and receive positive feedbacks on how to improve or even discard some ideas.

Job Submission

- A scientist prepares input data files, job scripts and application for many jobs.
- He then transfers them to the data grid.

- The jobs are submitted to the Grid infrastructure from users desktop with the jobs handling metascheduling, data transfer between the compute resource and data grid, and automatic collection of metadata.
- The user monitors the progress of jobs from their desktop.
- When jobs are completed, the scientist views the output files using XML tools and collates the core result from the metadata database.
- The data grid provides a data staging tool that integrates easily into the users desktop (the SRB provides something like a file system view of their data) and provides them with complete control of their files.

The User Interface

The primary user interface to job submission is the MCS (`my_condor_submit`) tool. The MCS is a perl program that uses the Condor-G wrapping of Globus together with a simple workflow (based on Condor 's DAGman tool) to incorporate data management. MCS requires that the user prepare a job description based on standard condor scripts. It is designed to work with the SRB. The MCS executes jobs in the following manner:

- users place all their input files into a specified directory.
- On execution MCS downloads all the relevant files (input files and executables) from the SRB to the Globus gatekeeper for the grid computing resources being used (by submitting a perl script `pre.pl`).
- MCS then run the job through the remote job management system using the downloaded files as input for the calculation.
- At completion of job the MCS will run another script (`post.pl`) which moves the generated files from the remote to the SRB.

There currently exists a web-service that allows MCS to be launched from a computer that does not have either Globus or Condor tools installed (figure 2.5).

2.4.2.2 Ganga, A user interface for Atlas and LHCb

Ganga (GAUDI/ANTHENA and Grid Alliance) is a user interface that gives access both to local resources and to the Grid and provides job-configuration and data management tools matched to the computing environments of the particle-physics experiments, Atlas and LHCb [Brook, N et al, 2003]. Atlas (A Toroidal LHC ApparatuS) and LHCb (Large Hadron Collider beauty) are two of the six particle

physics detector experiments built on the Large Hadron Collider accelerator. The interface is designed as a front end for handling framework-based jobs to the GAUDI/ATHENA software framework used by the LHCb and Atlas. This software supports full spectrum of data-processing applications including simulation, reconstruction and physics analysis. This interface is built for a group of physicists performing experiments that require analysis of data volumes in the order of petabytes per year. This Data is distributed between multiple locations for simulations carried out by the physicists to generate an understanding of detector behaviour.

Ganga relies on middleware like Globus and EDG (European Data Grid) [64] to perform its grid operations and it covers all phases of a job cycle: creation, configuration, splitting and reassembly, script generation, file transfer to and from worker nodes, submission, run-time setup, monitoring, and reporting. Ganga is implemented in python and follows a component-architecture using an object-oriented approach. Component-based approach means that it has the advantage that it allows reuse of components from other systems that are architecturally similar. Ganga components can be accessed through the command line interface and through a GUI (Graphical user interface). The components are divided into three: general, domain specific and external.

The general component deals with job definition component which characterises a GANGA job in the following terms:

- A job-registry component for storage and recovery of information for job objects and allow job objects to be serialised.
- Script-generation component that translates a jobs workflow into python instructions to be executed when a job is run.
- A job-submission component for submitting workflow scripts to a destination indicated by user, creating a JDL (Job description language) files where necessary, and translating the requests into format expected by the target system (EDG, local PBS queue, GridPP Grid and so on).
- File-transfer component to handle transfers between sites of job input and output file (involves adding appropriate commands to the workflow script at the time of job submission).

- A job monitoring component to keep track of job status and allow user to initiate and schedule queries.

The domain specific components are optimised for the specific GAUDI job for the Atlas and LHCb project. This provides the following:

- A workflow template covering a variety of common tasks such as simulating events and analysing some data sets.
- A component for GAUDI job-options editing, allowing selection of the algorithms to be run and modification of the properties.
- A component to allow large jobs to be broken down in smaller jobs e.g. examining the list of input files data files and creating jobs for subset of these files.
- A component for output collection and even merges outputs from sub jobs.

The external component includes modules of python standard library and non-python components for which an appropriate interface has been written.

Functionality

Ganga provides tools implemented for a wide range of tasks but with more focus on running one type of job for Atlas (ALTFast) and one type of job for LHCb (DAVINCI). The following have been implemented to make the user's job easier.

- Creation of JDL files necessary for job submission to the EDG testbed and generation of scripts to submit jobs to other batch systems has been automated.
- Most parameters relevant for ATLFast and DAVINCI have defaults values in GANGA so a user only supplies minimal information to create a job. Existing jobs can be created deleted or even edited by user in a template.
- A generic job-splitting mechanism exists where a splitter function is used to specify the way in which sub-jobs differ from initial job set up by the user.
- When jobs are submitted, Ganga starts to monitor the job state by periodically querying the computing system.
- When a job is completed, the output automatically is transferred to a dedicated directory or other location specified for job output files.

Job-Handling

The job registry keeps metadata about the users' jobs. It acts as a control centre for creation, configuration, submission, termination and monitoring of jobs. Jobs are represented as objects that include information about the job status, associated workflow and computing resources required. A job handler takes care of specific steps for job submission and monitoring. For submission, the handler parses the job resource requirements and translates it to system specific commands. For monitoring, the handler returns system-dependent information about the job status. GANGA has components containing job handlers to work with the local computer, a local PBS or LSF batch system and the EDG testbed.

A workflow is implemented that allows for the specification of the application to be run (executable or script), configuration parameters, and input and output files. For transfer of input and output files to and from worker nodes, GANGA uses the local system copy command, the gridftp transfer protocol and the EDG sandbox mechanism.

Graphical User Interface

The GANGA GUI is based on WXPYTHON, the extension module that embeds the WXWindows platform-independent application framework. The design of the GUI is based on a mapping of major ganga core classes – jobs, executables, files etc – onto corresponding GUI classes. The GUI is divided into three windows, the left section displays the job tree; the right section displays a variety of panels where user input is given, for example job setup; the lower section hosts an embedded python shell and doubles as a log window.

The job tree shows all job related values and parameters. The most important value is brought to the top and least hidden. There are five folders in the Job tree namely; Jobs, Configured, Submitted, Running and Completed. When the monitoring service is switched on, depending on the state of the job, it automatically moves to the appropriate folder. The monitoring service runs its own thread to avoid delays to users input. The Jobs folder is where new jobs are created and configured. Actions on a job can be performed from the menu bar.

2.4.2.3 Styx Grid Services (SGS)

The SGS system [92] is a means for running applications remotely as though they were installed locally. The software is written in java. To use the SGS system to submit jobs you need to deploy the program as SGS Service and all the other processes are handled automatically. The SGS system runs jobs on Grid systems like Condor or SGE (Sun Grid Engine) by making changes to the configuration file. Plugins' are created to allow the server talk to the underlying Grid environment.

Job Submission

This is best illustrated with the following example: A simple visualization program called *makepic* that reads an input file and creates a visualization of the results as a PNG. The *makepic* program is installed on the SGS server. A simple XML configuration file is created that describes the program in terms of its inputs, outputs and command-line arguments. The SGS server program is then started.

Clients can then run the *makepic* SGS service from remote locations as if the program was deployed on their local machine. They do this using the *SGSRun* program, a generic program for running SGS service:

SGRRun <hostname> <port> \makepic -I input.dat -o pic.png (<hostname> is name or ip address and <port> is port number of SGS server respectively).

A shell script (*makepic*) is written to reduce the command the user types by wrapping the *SGRRun* program with the hostname and port so the user just needs to run:

Makepic -i input.dat -o pic.png. The following steps then occurs:

- User initiates the above command.
- The *SGSRun* program connects to the server and downloads the XML description of the *makepic* program.
- It runs this configuration information to parse the command line arguments that the user has provided. It knows the *input.dat* is an input file and uploads it automatically from the user's machine to the SGS server before the program is started.
- When program starts, the Server knows that an output file, *pic.png* would be produced which it then downloads to the users machine.

- When user needs to execute an executable several time over input files, usually known as “high throughput computing” a slight change is made to how command is issued. User stores all the input files for example in a directory called inputs and then issue command *Makepic -i inputs -o pngfiles*
- When the SGS server sees that a directory exists instead of a single file, it sees this as a cue to run the makepic program over each file in the input directory, producing a picture for each file and placing them in a directory called pngfiles on the users’ local machine. The server uses the underlying Grid system like Condor to run these tasks in parallel on the worker nodes. The user interface is via the command line.

2.5 Requirements derived from Existing projects

A summary of the all Grid projects is presented in table 2.2. The Grid workflow projects have been reviewed and some general conclusions can be drawn. Geodise, ^myGrid and Dame have implemented the portal approach. Triana has used the application-based approach. Also all but Triana are targeted towards a specific user environment. The workflow Grid projects showed how the Grid can be programmed successfully using workflows. The target users are interested in using this tool for their daily activities. The user community consist of users that have vast knowledge in programming and would not mind programming the Grid. This implies that users have to learn new tools and ways to Grid-enable their application onto the Grid. These projects are more geared towards demonstrating the functionality of the Grid rather than the usability and as such the usability aspects of the interface have been neglected.

The user friendly projects have tried to make the Grid more usable by making improvements on the Grid workflow projects. These projects have tried to accommodate not just experienced users but users with limited knowledge.

Name of Grid Project	Deployment:	Integration with user's existing code.	Job submission process	Target User base
MyGrid	Portal	Users publish code to web and have to learn services	Workflow based	Biologists
Triana	Application	Users code has to work with a workflow description language	Workflow based	Problem solving environment for users working with wave signals
Geodise	Portal	Users code must be in Matlab	Workflow based	Engineers who already program in Matlab
Dame	Both	Users presented with a tool that they have to learn.	Workflow based	Users in an aircraft environment
eMineral	Portal	Interface wrapped around users existing code	Workflow based(Standard condor scripts)	Users comfortable with shell tools.
Ganga	Both	Users codes mapped into GUI classes for ease of use.	Workflow based	Physicists
Styx Grid Services(SGS)	Portal	Program has to be deployed as SGS Grid service but works with existing user codes.	Application based	Users that run applications remotely.

Table 2.2: Summary of the Grid projects reviewed.

These have been implemented but still lack features (such as an interface that can accommodate all categories of users, an interface that can accommodate users programming style) that is necessary to make the Grid easier to use. These user friendly projects have all made attempts to improve usability of the Grid but the key characteristic is that they still expect the user to change their programming style to use the Grid. Some of the interfaces are via the 'command line' which is targeted

toward the Linux user community comfortable with the style. The users not familiar with using the command line to communicate to remote systems will struggle.

In all the projects looked at, it is still obvious that the procedure to Grid-enable a new application on the Grid may prove useful and difficult and it requires that the user will have to either implement their applications using specific languages or express their code in a way that can fit into the Grid domain being offered to them. It suffices to say here that review of these projects and the knowledge gained has been invaluable in the design of the Grid user interface that will accommodate a users existing programming style.

The review of these projects has allowed us to highlight features that are important in implementing a grid enabled environment. These features have helped in determining some of the requirements of a Grid user. In reviewing the projects it was evident that the usability aspects of the project were not a priority. The focus was more on the functionality and what the system had to offer users. Unfortunately, users are not able to see the usefulness of a product if they cannot interact with it in an efficient manner. Most of these Grid projects really do explore the vast opportunities offered by the Grid but are not able to appeal to the number of users usually needed to guarantee their longevity. We looked at ways that these Grid projects could make the usability of the Grid easier; we believe that more time spent on training these users to use the interface could make a tremendous impact on how they embrace the Grid. We also believe proper documentation of how the interface works would be very useful

In this project we aim to explore the feasibility of implementing a piece of software that can be compatible with any user's working environment with focus on the usability aspects of the Grid. We believe that if users are presented with an interface that is close to a conceptual view of their own working domain, we can engage their interest permanently. This goal has made us design the interface based on the requirements discussed in the preceding section.

2.6 Conclusion

The literature review has looked at all the important aspects of the Grid and the necessary tools that are needed to program the grid. The Grid was defined and the history of other distributed technologies that existed before the Grid has been discussed. The various projects reviewed showed how the Grid can be programmed with the use of workflows.

The choice of Grid middleware was influenced by other projects that had used it. It is also important to note that all except the Triana and MyGrid Project used the Globus Toolkit for their middleware. This then shows the importance of this toolkit and how it has helped developers in implementing their projects.

This project is very concerned with how to deploy a software/code/application on the Grid in a flexible manner. To achieve this we have concerned ourselves with making sure we built an interface that is a user-centred. We have also looked at the possibility of making sure that the remote Grid network offering the computational resources to the users can cope with the demand and have most of the resources needed for a successful execution of a users' code. The usability aspects of the project were also thoroughly reviewed so as to make sure that the users requirements are met and satisfied.

3 System Design and implementation

The main goal of this project is to allow users to have access to a Grid-enabled environment that is simple and easy to manipulate. Users must feel very comfortable in this environment whilst working in the domain that they are already used to. A Grid-enabled environment can be deployed using two main approaches; either via a portal approach (users use a web browser to go to a URL and access the service) or application approach (software installed on client computer through which the Grid-enabled environment can be accessed). The Application-based approach is the area that mainly concerns the work here, as it allows interaction with the existing applications and data on the user's machine. This approach would allow us present to a user an interface that allows him to integrate his existing application easily with the Grid. Some users would want to use Grid interfaces without having to change the application style they are used to. The application-based approach would allow us integrate an existing user application with the Grid environment.

The Globus toolkit has become very essential for building a useful and efficient Grid environment and was the middleware chosen by most of the e-Science projects looked at the chapter two. Some of the Grid implementations have not been dependent on this toolkit but rather implemented Grid Services that can be accessed as web services. Some that have been implemented using the Application based have been developed using Java and do not need the Globus toolkit to function. It is therefore important to note that the way the Grid environment is implemented does not matter in as much as the user requirement is met. This toolkit was used in this project to explore building a piece of software that can allow a user to efficiently submit jobs on the Grid.

In this section we look at the exploring the user community, establishing user requirements and issues encountered during the design process. The tools for implementing the interface have been investigated and the user interface design was produced for implementation of the software.

3.1 Design methodology for the project

Careful study of the methodologies in the literature review showed that a bit of each was needed for this research. Each of these methodologies requires that some requirements are met for proper implementation and whichever requirement was met by the project was used to achieve the specific goal. This project aims to design a new interface to access a computational Grid and as such it required that users were given a different interface to the way in which they would normally access the Grid. Because we are aware of the usability problems that users currently encounter in using the Grid, we hope that this interface can help to improve their current usability of the Grid.

For the Grid application demonstrator, the waterfall model provided the structure needed to build the software and some of the criteria needed to build it. The diversity of the users interviewed meant that a prototype had to be built to demonstrate and show the users what the interface is trying to achieve.

It was possible to speak to three users of the Grid which is documented in chapter four. Interviewing the users really did help to understand how people currently use the Grid and to see how the new software could improve their current usability of the Grid. It also helped to capture current usability and functional requirements and see how the new interface could improve it. So invariably some steps in each methodology were used in the design of this system. The six steps taken in the design of the system are:

1. Requirement analysis and definition(Classic software cycle)
2. System and software design(Classic software cycle)
3. Implementation and unit testing(Classic software cycle)
4. Limited functionality simulation(Iterative design and prototyping)
5. Heuristic evaluation(Usability engineering)
6. Results/Feedback

3.2 Exploring the user Community

This is a very important aspect of this project as it is necessary to identify the various types of users and what their needs are at any point in time. Users differ and users have different expectations of the same system. Dr Tom Harris who looked at four companies from the media, Engineering, Services and Architectural sectors conducted an interview to explore the potential of a Grid computing based Digital Service Factory (Harris T, 2004). The report gives an insight to the various requirements and expectation of an average Grid user

3.2.1 Issues in Usability evaluation of the Grid System

In order to evaluate the usability of a system we need to identify the issues, on which usability hinges. We need to consider the needs and the characteristics of the users that are going to use the system. Specify their tasks, their goals and finally find out what they expect from the system. We address the most fundamental issues that concern a particular system, in our case Grid-enabled environment, that are important for its usability.

The method considered for evaluating the usability of the system is Heuristic evaluation (Nielsen and Mack, 1994). This is a usability engineering method for finding the usability problems in a user interface design so that they can be addressed as part of an interactive design process. This evaluation method involves having a small set of evaluators say about five, to conduct the evaluation of the user interface. Most times this is done with a prototype of the final system so as to allow for capturing the exact user expectation of the final system.

3.2.1.1 Consider Users Types

This software environment is aimed at supporting intermediate users typified of wanting to use the Grid without learning any special language. Consequently, it was also vital to take into account the different types of users that are available to use the interface. There are computer-experienced users for instance, that can easily adapt in the www world and make effective use of the system facilities. However, there are novice users that may be experts in their domain and not in the Grid domain; these users may require help to use the system effectively. Therefore we aimed to build a

system that allows for productive and expert work, while being simple enough to allow for easy exploration and learning.

3.2.1.2 Task analysis

The various tasks that could be demanded from users may be enormous. It would be very useful to capture tasks that can be performed by users of the Grid interface. The general task in this particular project is that a user would want to execute their compiled code successfully and have access to the result in a very convenient and efficient manner. The task considered for this interface is a user either submitting their source code for compilation before execution or a compiled code for execution. This would be provided with the following features: ease of use, accessibility and a user friendly interface.

There is a need to investigate the various day to day tasks that the stakeholders are involved in their present use of the Grid. This is necessary so we could incorporate this task in our proposed way of using the Grid and this would be used in determining the usefulness and efficiency of the system. The task involves a user submitting a source code or an already compiled code. Which ever task the user performs, we need to itemise this task and also what they would expect from the Grid user interface of the Grid-environment.

Tasks that the user could be involved in on a day to day basis

- Sending source codes for compilation.
- Sending Compiled codes with libraries needed to run it.
- Copying and moving data to and from the remote end.
- Monitoring jobs sent via the Grid interface.

3.3 Determining User Requirements

In the early part of this project an interview was conducted by Harris (Harris T, 2004). Four companies were interviewed and had different requirements and issues in using the Grid. This report was necessary to have a feel of the type of user community that were most likely going to embrace the Grid. Later in the project it was feasible to interview users of the White Rose Grid in the University of York due to the proximity and the convenience. There are many standard methods for determining the user

requirements. Of all the methods available the interview method was used. There are many users of the WRG and all were contacted but only three responded. These users from different scientific background were interviewed and the finding of each is stated below. Each of the interviews is described in the form of a use case to understand the way the three users presently interact with the Grid.

3.3.1 Use Case for User1

This particular user is an expert user of the Grid with a sound knowledge of the linux operating system. The users' daily job involves multithreading and uses shared memory so that his program can only run on one computer consisting of different processing nodes. His system uses the server client architecture where he can instruct the client to use for example four processing nodes on whatever computer is available. The user sees the Grid as an environment that provides him with as many nodes as possible to speed up the simulation of his experiment. The overall steps in which the user takes to achieve a successful job submission on the Grid is presented in the form of a use case below:

1. User set up a Bionc system that allows multiple executions of codes in a distributed environment.
2. The Bionc setup includes a Server in user local network and client installed on the Grid node.
3. Workunits are setup on the Bionc server with codes and the input on which they act on.
4. User logs on to the Grid.
5. User transfers compiled C code to the Grid node. User uses a library called *libxml2* which is also transferred to the Grid.
6. User then issues command `Qsub jobscript` to execute the job.
7. The script initiates the BioncClient.
8. The BioncClient connects to the BioncServer via an http connection on a particular port and retrieves the work unit and executes them accordingly.
9. The output data generated is transferred to the Bionc Server for user to view for further analysis.

3.3.2 Use Case for User2

This is a basic user that would have been particularly ideal for the interface being developed for this project. The user sees the Grid as a way to offload large scale simulation that takes a local machine a much longer time to execute. The user's code is usually in c++, python, perl, c-sharp and java and his primary operating system is windows but he uses linux for better interaction with the Grid. He currently has no way to monitor his job and he always redirects his standard output (Stdout) and standard error (Stderr) to his user directory on the Grid. If he has errors on his code, because his job is array-like, the Stderr always has the last error generated by the array. The users overall step in a successful job submission is itemised below in a use case:

1. The user populates a database with all the input parameters to run his job.
2. The user writes a c++ program which in this particular case is about three executables.
3. The user writes a BASH scripting file (pcscript).
4. The user copies the three exec file to an ftp server.
5. The user then *ssh* to the Bio Grid.
6. The user then types the command *Qsub pcscript* on command line.
7. The script then calls the sql file to get parameters.
8. The script downloads the executables to a local machine (e.g tmp directory).
9. The script runs a list of execs. The output of one exec serves as input to another.
10. The output is then sent to an ftp server.

3.3.3 Use Case for User3

This particular user is an expert user of the Grid. Average daily operations involve simulation that needs a large ram (random access memory) to run which can only be available on a Grid network. The Grid provides a way for him to speed up his simulation by half the time taken on his regular computer. This user's operating system is linux and he does not mind programming the Grid. The users overall step in a successful job submission is given below as a use case:

1. The user writes a parallel program in fortran90.
2. User has previously installed Castep code on the Grid node.

3. User transfers source code on the Grid node and compiles.
4. The source file for the job is a simple text file containing parameters needed by the castep software.
5. The user logs to the Grid node and issues the command, *Qsub <script file>*.
6. The output of the job is put on the root node of the processing node.
7. The output file generated is of two types. A text file which could be as large as 1mb and a binary file which could be as large as 1 gigabyte.

The conclusion that was drawn from the interview of the three users was that they all use the command line at the moment for accessing the Grid. User1 and user2 both use the Linux operating system while user2 uses the windows operating system. It was hoped that the only Grid user that used the windows operating system would play a major role in helping to determine the user requirements and probably help with giving useful advise in building and testing the interface but the user was not readily available and as such the interface was built based on how to improve the ways the three users currently use the grid

3.3.4 Users experiences in using the Grid.

The three users' ways of accessing the Grid has been explained in previous section. The problems and issues that they encountered in trying to Grid-enable these applications are further investigated. User3 experience particularly stands out as he had to go through the following steps in Grid-enabling his application:

- User3 had to move his software, castep to the White Rose Grid (WRG).
- This software needed a parallel debugger (Nevada) not available on the WRG and so that had to be installed.
- His source codes was written in fortan99 and mpi; a parallel programming language. When the user wrote his program serially and compiled on the Grid it was successful but had problems compiling and linking a parallel program due to library issues.
- The user then had to work with the WRG support group to manually install his software and update the necessary libraries needed before he could then successfully run his application

It is then obvious that libraries is one of the problems that users have when they want to Grid-enable their application and that is one of the problems that we have further investigated in this research and see how this could be tackled.

3.3.5 Requirements for the proposed system

Based on the Grid projects looked at and the above users way of using the Grid we have identified that users will expect to be presented with an interface that can integrate into their existing work domain and can accommodate their existing application/codes with minimal efforts from their part. Based on the requirements we are now able to present to users an interface that can aim to fulfil the following requirements:

- Users want to submit their existing code via interface without modification.
- The user wants to have their Job run successfully.
- The users want to recover from errors easily.
- User wants useful feedback and Monitoring throughout interaction.
- Users want icons that depict the actions to be performed.
- The interface will allow users perform drag and drop.
- User wants some Grid ‘commands’ represented with buttons on the interface.

3.4 Using the Grid

The previous section presented the way three particular users access the Grid. Generally the way people access the Grid using the White Rose Grid is via the command line. Regardless of the user’s primary operating system, users always follow a particular standard way for using the Grid. White Rose Grid users would normally apply for an account via a support engineer and the general steps taken for using the grid are as follows:

1. User originally applies to the white rose grid for a grid account to be able to log on for execution of job.
2. User then opens a command line interface and logs on the Grid.
3. User then writes a script that can simulate all the jobs to be run.
4. User then issues command to execute job e.g *QSub Script* for a sun Grid engine machine or *globusrun rslscript* for a linux machine running Globus.
5. Whatever results are either stored in the remote end or transferred to user.

6. User then tries to retrieve the result and log out.

3.4.1 Proposed way of using the Grid

It is hoped that the user's credentials for logging to the Grid is stored on the local end so that user can from the interface click a button for the log on. User can then chose a button to either submit a source code to be compiled or a compiled code to be executed on the remote end. The user then has useful feedback on the progress of the job. The interface would be useful for users who are not so familiar with the command line way of interacting. In particular, one of the users stated that it would be useful to click a button from the interface to issue the *Qsub* command for submitting a job and have the result displayed in the users' workspace or the data repository of the interface.

3.5 Grid-enabling an application

When a user application has been running standalone on a workstation, the user assumes a particular way of working peculiar to their domain. When this application needs to be moved to the Grid due to resources not provided by the user's domain, the user's interaction to the application may change slightly when he now needs to use it on the Grid. The architecture of the interface is presented below and the stages that will occur in Grid-enabling an application is also presented.

3.5.1 Grid Interface Architecture

The Grid interface architecture during implementation consisted of the Grid User interface, the client cog kit, the underlying library verification perl programs, the samba client service and the remote Grid network to which communication is made every time a user wants to submit a job. Figure 3.1 shows the interface architecture from when the user logs on to the remote Grid network to when the user jobs is completed.

3.5.2 Job Description

This particular step involves how a user would describe his job to be submitted on to the Grid. Every programmer regardless of language used must have an executable which they want to execute. This executable could produce several inputs and outputs

and as such we must explore how we can handle this situation. The Globus middleware does support some job description languages for which this complicated job can be described. The middleware also supports running of scripts which some users use to describe how their jobs can be run. We could also have users with an executable that would require input from another executable to run. Which ever scenario that exists we are concerned with how the interface can accommodate this. A typical job will consist of a user either trying to submit their source code/codes to the Grid for compilation or their already compiled code with dependent libraries to the Grid. In this project we have looked at the scenario of an executable and an input file for a successful job submission. This interface can be further extended to cater for other scenarios.

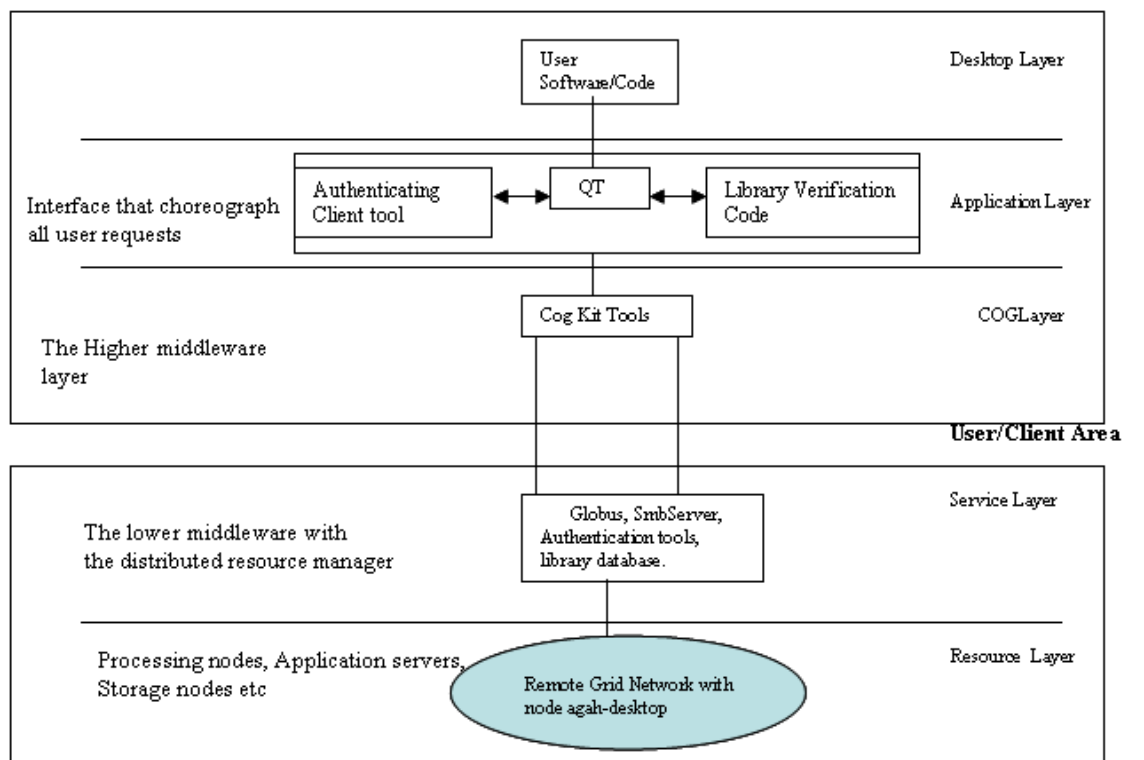


Fig 3.1: Grid Interface Architecture

Underlying Grid Network Area

3.5.3 Grid interface checks if request can be handled

This section would have a list of different source codes that can be compiled as well as the compiled codes that can successfully run on the Grid. Anytime a user wants to run a job this process must make sure that all the necessary checks are carried to ascertain that user’s application requirement can be catered for by the Grid interface.

This is one important aspect of the user's requirements as users would be more comfortable with an interface that can accommodate their existing codes.

The Grid Interface must mirror the Meta data information kept by the GRIS (Grid Resource Information System). GRIS is the information provider component of Globus that has the list of Processing nodes and what they support (graphical illustration is depicted in table 3.1). This information is very important as this is needed by the interface to correctly support a user's existing codes and whenever the Grid environment can handle more applications, the Grid interface is then able to cater for users with those types of applications.

Name of GridMiddleware: Globus

Type of Job Description Language: JSL, JDL, WSFL

Type of Applications supported: java, c++ and matlab executable.

List of Processing Nodes:

Node Name	O/S	Compiler Information	No of Processors	MEM	Application	Database
Pascalli	Linux	JVM, MathLab, Gcc	8	24gb,	MPI,	OGSA- DAI
Maxima	Linux	JVM, Gcc	12	40gb	MPI	Oracle
Tatania	Linux	Mathlab, Fortran99	14	50gb	MPI	Oracle
Iceberg	Linux	Gcc, JVM	24	30gb	MPI	SQL

Table 3.1: Sample table of information about processing nodes

The big picture of this Grid interface would be to have access to many Grid environments and present to users a number of Grid services so that based on users budget, a user may decide to go for a service that would process his task faster but would cost him or a service that would be slow but can guarantee that his final output is acceptable to him (Quality of service guaranteed). The following table is an example of the kind of information that should be available to the Grid user interface to help match a user's requirement to the processing nodes available to handle a job request.

This information is used to check against the user application before going ahead with the submission. This stage just checks that there exist enough resources that can do the job but may not be able to guarantee that they are readily available for the job.

3.5.4 The Request is verified

When the Table verifies that the user application can be handled, then the next stage would be to verify that there exist resources to execute the job. This stage involves the Grid interface checking with the underlying Grid middleware if that Job can be handled. If this is possible, there is some information that would be useful to the user at this stage. Here the user may be interested in knowing that his job can be submitted but there exist a lot of queues and may take a while for his job to be submitted and if so it is hard to tell when it can be completed. At this stage the user can decide to forget about the job and execute it locally or opt for an expensive option of going to a provider that has readily accessible processing nodes waiting to execute the job. This stage also handles the library verification process explained in the previous section and makes sure that the libraries are properly updated and moved to the remote end prior to the Job being executed.

3.5.5 Job is submitted to the underlying Grid environment

This particular step involves three major entities; the application, the processing nodes, and the data repository. The different ways in which this can be achieved is affected by the type of application that is being submitted, the bandwidth requirement, security, data size etc. This section is also concerned with the ways in which the Schedulers submit the jobs. So this is divided into two parts; Application Execution and Scheduling Job Execution. When Job is submitted, there must be a feedback of how long it will take for the job to be executed. The factors that affect the effective completion of a job and questions that would be asked are listed below:

- The type of Operating system the users application runs on
- The Grid user interface designed by this project
- The Cog Toolkit
- The Grid Middleware(Globus)

- The Grid Resources
 - Are they available?
 - Are they free for use?
 - What is their processing speed?
 - Load balancing and sharing policies used.

3.5.5.1 Application Execution

Different scenarios of this are stated below. The three standard steps for an application to be executed are initiating the application, executing the application and the final output of the application. One or more processing nodes could perform these three steps. Depending on the amount of processing nodes to perform the operation, a lot of issues are considered. When a processing node is required to perform this operation then it is the simplest of all scenarios and straightforward. But when you need more than one processing node then you have to consider other factors that affect successful execution of the Job such as speed, bandwidth reliability etc. The type of application to be executed also needs to be considered; as seen in the Grid user interviewed, users application can differ. Some applications are standalone executables while some need other executables and may need to run in a parallel mode to benefit from the Grid. The processing node therefore acts as a workstation, server and a store. In a Grid environment the workstation would be at the client end whilst the server and store would be in the Grid network, which may require a local area network or wide area network connectivity to be accessed. Based on this, the following scenario of how a job could be initiated is discussed below.

1. The job can be an executable that needs to be transferred to the Grid network together with all the necessary dependent libraries. We then access the input data needed by the executable and decide where the output is be stored.
2. It can be a Java executable in which case we would require that a JVM (Java Virtual machine) is at the Processing nodes for the execution to take place. This would also require a descriptive way to tell the underlying Grid infrastructure how to execute the job.
3. It could be that the application already exists on the remote Grid network and all you need is a way to run the application after successful logon to the Grid network.

4. Another scenario could be that the application is transferred to a vendor store and when the Job is issued, the transfer takes place from that store to the Grid network. In cases like this, the vendor store may have a better communication medium to the Grid network than the user location.
5. A scenario can be to have a Proxy server to the Grid middleware. The Proxy server acts as a gateway between the application and Grid Environment and resides at the application local network but gives full access to the Grid middleware.
6. It can also be that the user application is on the same platform as the underlying Grid middleware and the user has a front end to the Grid middleware in which case no further action is needed for its execution.

3.5.5.2 Scheduling Job Execution

Scheduling in a Grid environment is quite a difficult task as the system is expected to cope with different types of applications and specific requirements of these applications. Scheduling is the term that refers to how processes are assigned to run on different computers especially for cases where there are more processes than processing nodes available. This is usually handled by the Scheduler. There are two major types of scheduling: local and Global. Local scheduling describes how a scheduler allocates processes to the local domain or network that it manages. Global scheduling involves a master scheduler having to control other local schedulers of different domains that are coupled together. The one of interest to us is the Global and this has been easy to achieve in tightly coupled systems. Global Scheduling is hard in loosely coupled system due to the heterogeneous nature of the features peculiar to each one of them. So the underlying Grid environment could decide to make sure the processing nodes for executing tasks are tightly coupled or go for a loosely coupled system where all nodes have the same operating system etc. Each decision made has its advantages and disadvantages. Presently most Grid networks consist of nodes that run Linux and Sun Grid engine as their operating system and it may be difficult to cope with windows specific application. So it is really a matter of trade offs. The tightly coupled system would allow for easy scheduling while the loosely coupled would allow for a wider range of application execution. This system aims to allow you take advantage of all the computers available to you to make your Grid network. It would be advisable to build into the job submission technique, the

querying of resource utilisation in order to achieve load balancing across the resources responsible for processing tasks. (Bessis et al 2006) have looked at home intelligent optimisation techniques that can improve job scheduling in a Grid environment. The paper is a review of old techniques, identifying the problem and coming up with a proposal. We can then say that the response time of a job submitted to the grid will be: Time for External Scheduler to submit job + Time for local scheduler to queue the job + Time for Actual resource to perform the job + Time for resource to send the result back to user or to a storage indicated by user for later viewing (Bessis et al 2006). For the purpose of this project, the external scheduler would not be used, as the Grid user interface would be interfacing with the Grid network that would guarantee scheduling and provide the necessary computational and storage resources.

3.5.5.3 Application Specific requirements

The type of application that is being submitted is important. Applications can be classified broadly as Data dependent and Processor dependent. If an application is data dependent it may not care how long it takes for a job to be completed. All that matters is the quality of results. Such application can be on the queue and just needs to know when their deadline can be met. The Grid environment must then be concerned with making sure that the error recovery and transmission issues are properly addressed as these can in a large extent affect the quality of result produced. These types of application are also concerned about trust and privacy issues to guarantee that their data are protected from malicious acts and unauthorized use. Some application are processor i/o dependent in which case they are interested in how long it would take to complete their execution and if their deadline can be met. In this case the underlying Grid middleware would have to guarantee the resources to execute the job. Some applications are database inclined and would like to know that the underlying database infrastructure could handle the data formats supported by their application use. These three issues (Deadline issues, Quality of result, Data issues) are reliant on the underlying Grid network responsible for the Job execution.

3.5.6 Job Monitoring

There are various ways of monitoring jobs in traditional non-distributed systems. This is usually done by the underlying operating system. For example in Unix when you submit a job it can give useful information like the ProcessId of a job, the ProcessId

of the parent job, CPU usage and memory consumption of each job. In windows, there exist graphical tools that can be used to monitor jobs. In Grid systems there exist some job monitoring tools which are used in a homogeneous Grid environment. The Load Sharing Facility (LSF) [75] is used for resource management for super computers. It has a lot of functions including providing information about a job status and job accounting. The Portable Batch system (PBS) [87] offers the flexibility to schedule jobs and allow users to start and terminate jobs as required. The Distributed resource management application API (drmaa) [61] aims to develop an API specification for the submission and control of jobs to one or more distributed resource management system such as condor, Globus and Sungrid systems. As the underlying middleware would be Globus, users' jobs can be monitored using the drmaa version for Globus. To keep the interface simple, once a user submits a job, an indicator on the Status bar will show to the user how long it will take for his job to complete. This cannot be represented in number terms as this is usually unpredictable due to a number of factors that can hinder how long it takes to complete the job.

3.5.7 Result Returned to user

Depending on users choice during the job submission results can be returned in a number of ways. A user may specify that he wants his job returned to his local working directory or may require a storage space on the Grid system to store his result for future use. The Grid user interface would aim to deal with whatever options the user chooses in waiting for his result.

3.6 Deployment

There are a lot of issues that have to be considered in the implementation of this project. Firstly the target users of this project are identified and their requirements captured. These requirements would enable us to set out criteria for determining the success or failure of the implementation. As stated in earlier chapters a semi – functional prototype of the software has been produced and tested amongst Grid users. Most Grid implementation can be deployed in two main categories namely Application or Portal. There is really no best approach to deploying a Grid-enabled environment as the choice of deployment is usually based on the target user community. The application-based approach is chosen in this work and this has been

deployed by building software that is GUI based and presents a very simple and intuitive interface for a user to successfully submit a source code or a compiled code onto the Grid.

3.6.1 Different ways for Application-based deployment

There are many ways that this particular approach can be deployed with the underlying grid network. The different ways is explored in relation to how effectively, we can copy the libraries needed by either a source code or an already compiled code to the underlying Grid network. This is very important as it will help to fulfil the users requirements which is to allow users Grid-enable their application easily; compiled codes will not run correctly without the libraries needed to run them and so we must be able to determine how to successfully transfer these libraries for use at the remote Grid environment.. With either of these two options, we must investigate how we can deal with libraries that are needed for the proper compilation of the source code or libraries that are needed for the proper execution for a compiled code. Each one has its drawbacks and advantages but we must present to the user these two options for flexibility.

3.6.1.1 Copy Libraries to Resource provider

This option involves copying the libraries needed by either the source or compiled code to the resource providers' library store to guarantee a successful run of the codes. Here the user has their source code on the client end and then need to copy both the executable/source code and libraries needed for successful execution to the standard library path for the remote; in this case */lib* or */usr/lib*. The resource provider would then need to allow the user, access to sensitive areas of the processing nodes where standard libraries are stored.

3.6.1.2 Copy Libraries to the user remote home drive

This option has minor risk. Here the user has their source code on the client end and then need to copy both the executable/source code and libraries needed for successful execution to the user remote home directory. With this option, the executables may not work and the users may have to do extra work to make it work. This option is the most versatile of all the others considered. With this option, we are able to explore different programming codes, find a way to handle libraries they need to execute their

codes and design a way that the remote end can cope with the requirements of the intended users.

3.6.1.3 Virtual Machine Deployment

This is the most risk free of all the options considered. With a virtual machine, we need not worry about library compatibility or system dependent issues. The draw back to this approach is that the size of the virtual machine would be too large for transfer over the network. The average size of a small virtual machine is 10 GB and even with a 8mbps bandwidth speed, it could take about 3 hrs to be transferred. It would have to be physically copied to say a storage device then taken to the Grid network to be run. The only software the Grid network would require would be the particular virtual machine server to run the virtual machine client.

3.6.1.4 Java virtual machine Deployment.

This option would require that a java virtual machine (JVM) be installed on the remote end. The client then writes their java source which is then compiled by the javac compiler. The user can then send their java byte code to the remote end for execution. We would have to ensure in this case that the jvm versions of the client and remote are compatible with each other. But then not all users write in java so we must look for a deployment that can accommodate as many languages as possible.

These four options have been discussed above and each one can be implemented but due to the constraints that could be encountered for each one, the one with the least constraint was chosen. Copying Libraries directly to the resource provider is the most critical approach and can actually break the remote machine. Most service providers will not adhere to this option. The virtual machine option poses a transfer problem due to the large size of virtual machines. This JVM option is another viable option but we would be limited to source codes that are written in Java. The option to copy the source/compiled code libraries to the users remote location is the viable option for what this work aims to achieve. We can give users the opportunities to submit their source/compiled code, transfer necessary libraries for the successful execution of these codes and not worry about posing a threat to the remote processing nodes.

3.7 Issues in implementing the interface

In the course of designing the interface a lot of concerns were explored. These issues are discussed in the following sub section.

3.7.1 Computer Architecture and Design

Computer architecture is a term used to refer to the internal organization of any computer system. Computer architecture specifies the interconnection among major components and the overall functionality without giving many details that the architectural design omits (Comer, Douglas E. 2004). Computer architecture and design is a very broad topic and more reading can be explored in (Comer, Douglas E 2004). For the purpose of this project we are particular interested in the data and program representation because Grid environments consists of heterogeneous nodes and the way data and program are represented differ.

3.7.1.1 Data and program representation

This section explains how digital systems encode programs and data and how this underlying representation is important for a programmer writing software. The number of bits per byte is important to a programmer as computer programs use bytes to store values and the size can determine the maximum value that can be stored. A byte that contains k bits can represent one of 2^k values. Subsequently an 8 bit byte can represent 256 possible values. The Hexadecimal notation is popular for conversion between the binary and hexadecimal since sixteen is a power of two. A more in-depth of this topic can be found in (Comer, Douglas E, 2004).

Numbering of bits and bytes is very important for the transfer of data between one computer and the other. The question then, is how should bits and bytes be numbered? When sending a byte of data over a network both the sending and receiving computers must agree on whether the least significant bit or most significant bit is transferred first. (Comer, Douglas E, 2004). *Little endian* characterizes a system that numbers bytes of an integer from least significant to most significant and *big endian* characterize a system that numbers bytes of an integer from most-significant to least significant(illustration in listing 3.1).

```

// Operation:
// 1. Copy from client to host
// 2. Cause it to be compiled
// 3. Run it and get the result back (standard out)
//
// Result: on little endian - 255, on big endian - 16711680
//
#include <iostream>
using namespace std;
int main(int argc, char ** argv) {
    unsigned short shorts[] = {255, 0};
    int * n = (int *)shorts;
    cout << *n << endl;
    return 0;
}

```

Listing 3.1: Determining the endianness of a computer

In a Grid environment, most computers do not have a prior knowledge about their endianness and as such it is necessary to find a way to circumvent. The program below illustrates how we can determine the endianness of a computer before we transfer data to it. We then find a robust way of determining the endianness of a computer and listing 3.1 will return the correct value of 255 stored in the array if the system is little endian and otherwise if not. The program was run on two systems that returned the value 255 indicating little endian and 16711680 indicating big endian.

This program is run from the remote end and sent to the client every time a user needs to execute a job so that if the endian is not the same then a swap can be made to make sure data it is stored correctly and when retrieved, the value is what we expect. This byte swap depends on the length of the variable stored in the file because a 2 byte integer file requires a different swap than a 4 byte integer swap. We then conclude that it is not possible to have a general program to convert the endian in a binary file and for the purpose of this research, our test computers all have the little endian.

It would be good to find a general method that can solve this problem as it is very critical in the way data is represented on computers. Other alternatives to solving this problem would be if the use of virtual machines is introduced. With virtual machines you really don't have to worry about the endianness but it brings other problems of the

size and time taken to transfer on the communication link between the user and the Grid-enabled environment.

3.7.2 Library issues

Libraries deal with additional information that a computer program or code needs in order to complete its execution. Different computer platforms have distinct way of storing libraries. There exist a lot of platforms, and it would be difficult to consider all of these. The windows and Linux platform is discussed here, as they are the most popular. In Windows, most libraries are stored in the *C:\Windows\system* or *C:\Windows\system32* directory. In Linux, the libraries are stored in */usr/lib* and */lib* location of the root directory. For this project a user can either submit a source code to the remote directory to be compiled or a compiled code to be executed. Whichever situation, we need to cater for how the libraries would be transferred to the remote end successfully without having duplicates that could overload the remote system.

3.7.2.1 Linkers and Loaders

Linking is the process of combining various pieces of code and data to form a single executable that can be loaded in memory. Linking can be at compile time, at load time (by loaders) and at run time (by application programs) [74]. Linking is done by the programming environment (IDE) by the use of a project. To further illustrate this, we use the ELF (executable and linking format) executables on the x86 architecture (*linux*) and the GNU compiler (*GCC*) and linker (*ld*). This illustration remains the same for a different processing architecture, operating system or object file format being used.

To understand how libraries operate, it is necessary to discuss the process of creating an executable code. When you write a c++ program for instance you always have a header file for example *a.h* and the main program for example *a.cpp*. The *a.cpp* would include the following lines in listing 3.2. Assuming we have two files *a.cpp* and *b.cpp* and corresponding header files *a.h* and *b.h*. We create a project say *example* and store these 4 files in it, then we run *gcc example* on the command prompt.

```
# include <iostream>
using namespace std;

include "a.h"
//example program.
```

Listing 3.2: Illustrating header files.

The first stage in compiling is the use of the pre-processor which executes before the compiler. In *a.cpp*, the `#include` command causes the pre-processor (*cpp* in linux) to cut and paste the header files *iostream.h* and *a.h* into the text/code that will be compiled. This makes the compiler compile all the code in *iostream.h* and *a.h* in addition to the user-written code in *a.cpp* and *b.cpp* to produce intermediate files *a.i* and *b.i*. To create the executable example program, there are two phases that takes place: compiling and linking [51].

Compiling involves taking the source codes in this case *a.cpp*, *b.cpp* (which contains definitions, functions, classes, constants and variables) and running compiler proper (*cc1* to produce *a.s* and *b.s* and then running an assembler (**as**) to generate the object file denoted with (**.obj** or **.o**) in this case *a.o* and *b.o*. This object file is the translation of the c++ source code into an architecture-dependent object code. Note that *cpp*, *cc1* and *as* are the GNUs' operating system pre-processor, compiler proper and assembler respectively that come with the *GCC* distribution. The linker then takes the two object files, *a.o* and *b.o* and produces the final executable in this case called *example*. This executable is then loaded in memory by the loader called *execve* which loads the code and data associated with it into the memory and runs the program by jumping to the first instruction.

The object file can come in three forms [74]:

- Relocatable object file, which contains binary code and data in a form that can be combined with other relocatable object files at compile time to create an executable object file.
- Executable object file, which contains binary code and data in a form that can be directly loaded into memory and executed.
- Shared object file, which is a special type of relocatable object file that can be loaded into memory and linked dynamically, either at load time or at run time.

3.7.2.2 Types of Program Libraries (x86 architecture Linux)

A program library is a file containing compiled code (and data) that is incorporated later into a program. Program libraries allow programs to be more modular, faster to recompile and easier to update [72]. Program libraries are divided into three types:

1. Static libraries

These are libraries that are installed into a program executable before the program can be run. They are collection of object files and end with the “.a” suffix and are created with the *ar* (archiver) program. To create a static library for example *my_library.a* with two object files *a.o* and *b.o* we issue the command *ar rcs my_library a.o b.o*. You can then use the library by invoking it as part of the compilation and linking process when creating the exec. With *GCC*, you use the *-l* option. Static libraries have bundled with the executable, all the libraries needed to run it. The drawback to this type of program library is that they are usually very large.

2. Shared Libraries

These are libraries that are loaded by programs when they start. When a shared library is installed properly, all programs that start afterwards automatically use it. With regards to shared libraries Linux permits the following:

- Update libraries and support programs that want to use older, non-backward compatible version of the libraries.
- Override specific libraries or specific functions in a library when executing a particular program.
- Do all this while the programs are running using existing libraries.

Shared libraries are able to support these properties based on a number of naming conventions and guidelines. So there is a difference between a library’s name, its “soname” and “realname” and how they interact.

Shared library names

Every library has a special name called the “soname”. The soname has the prefix, “lib”, the name of the library, the phrase “.so” followed by a period and a version no that is incremented whenever the interface changes. A fully qualified “soname” includes as a prefix the directory it’s in; on a working system it is simply a

symbolic link to the shared library's "real name". Every library has a 'real name' which is the file name containing the actual library code. The real name adds to the soname, a period, a minor number, another period and the release no. The last period and release number are optional but they help to tell you the versions of the library installed. The key to managing shared libraries is the separation of these names. When programs list internally the shared libraries they need they should only list the soname. When a new version of a library is installed, it is advisable to install it in a special directory and then run the program *ldconfig* which examines the existing files and creates the sonames as symbolic links to the real name as well as setting up the cache file */etc/ld.so.cache*. *Ldconfig* makes no assumptions about what programs to link to, so installers must specifically modify symbolic links to update what the linker will use for a library.

File System Placement

Shared libraries must be placed somewhere in the file system. Most open source software follow the GNU standards which recommends all libraries be installed in */usr/local/lib* when it is a distribution source code and all commands should go into */usr/local/bin*. The file system hierarchy (FSH) discusses what should go where in a distribution. According to the FSH, most libraries should be installed in */usr/lib* but libraries required for start-up should be in the */lib* and libraries not part of the system, should be in */usr/local/lib*. Most systems automatically check the */usr/local/lib* as default for their search for libraries

3. Dynamically linked (DL) libraries

These are libraries that are loaded at times other than during the start-up of a program. They are useful for implementing plugins and modules because they can stall the loading of the plugin until it is needed. DL are useful for implementing interpreters that wish to occasionally compile their code into machine code and then use the compiled version for efficiency purposes without stopping the program running [74]. To implement a DL library in Linux, you write an API for opening the library, looking up symbols, handling errors and closing the library. A programmer would have to include the header file `<dlfcn.h>` to use the API and the "dlopen()" API is used for the interface.

3.7.2.3 How libraries are used (x86 architecture Linux)

On GNU based system including all Linux system, starting an Elf binary executable automatically causes the program loader to be loaded and run. On linux this loader is named */lib/ld-linux.so.X* (where X is a version no). This loader then finds and loads all other shared libraries used by the program. `LD_LIBRARY_PATH` is a set of directories where libraries will be searched for first by the loader before the standard set of directories. `LD_PRELOAD` lists shared libraries with functions that override the standard set and are implemented by */lib/ld-linux.so*. Unfortunately `LD_LIBRARY_PATH` is not a standard for all Unix-like systems. This same functionality is available on HP-UX the environment variable as `SHLIB_PATH` and on AIX, this same functionality is through the variable `LIBPATH` (with the same syntax) [72].

The standard set of directories is the next place the loader will search for libraries and it is stored in the file */etc/ld.so.conf* and you could manually add other directories to be searched in this file. Searching all directories can be ineffective so a caching arrangement is used. The program *ldconfig* by default reads in the file */etc/ld.so.conf*, sets up the appropriate symbolic links in the dynamic link directories and then writes a cache to */etc/ld.so.cache* that is then used by other programs [72]. This indicates that *ldconfig* is run whenever a *dll* is added, removed or the DL directory changes.

3.7.2.4 Deploying an Application

This particular process was necessary to see how an application built in one system could be deployed to another system using the same operating system. This is further illustrated using a QT (A cross platform application framework for developing GUIs?) application built on one windows machine and deployed on another windows machine. When you build an application you end up with an executable and there are two ways to deploy the application:

1. As a bundle(copy the executable and libraries needed to run it and transfer to the target machine)
2. Through the installer (A software package that contains all the information needed to run the application. The programmer authors the installer and the target machine can run it and install it.

The second method is a straight forward way and useful for commercial purposes but not useful for Grid environments. The first option is useful in Grid environment.

We built the application and we discovered that we needed to have the executable, QT libraries and Compilers specific libraries to successfully run the application. We used the “depends” tool in windows to know which libraries were needed. We then needed to decide whether to deploy the application statically or dynamically. Both options worked but the choice taken must be based on how the application is built. A table highlighting the features is presented in table 3.2.

STATIC LINKING	DYNAMIC LINKING
Results in stand-alone executable	Results in an executable with some dynamic link libraries
There are a few files to deploy	There are more files to deploy
The executable is large and with no flexibility. For example a new version of the application or build software would require that the development process is repeated.	Executable is smaller and flexible. User can independently upgrade libraries used by the application.
You cannot deploy plugins.	Can be used to deploy plug-in based applications.
Good if the libraries are only going to be needed by the application.	Good if you are using libraries for a family of applications

Table 3.2: Comparison of Static and Dynamic Linking

With this comparison, it was easy to draw a conclusion that the dynamic linking deployment would be supported by the Grid user interface. This is because static linking would result in a big executable which may be too big for transfer. Also the types of users that use the Grid are ever changing their codes and the static linking may prove too cumbersome and frustrating. The process of achieving this method of deployment in a Grid environment is then further explored in the next section.

3.7.2.5 Running the executable from the remote end

The previous section discusses extensively the procedure that takes place when a program /code/application/executable are run on any computer. We then need to investigate how we can accomplish this same procedure remotely. For the purpose of this research the computer used in the remote grid network is an x86 architecture (Linux) and so the process of how an executable is run, is discussed based on this architecture.

There are other ways an executable can be run remotely and this is discussed below:

1. The job can be an executable that needs to be transferred to the Grid network together with all the necessary dependent libraries. We then access the input data needed by the executable and decide where the output is to be stored.
2. It can be a Java executable in which case we would require that a JVM (Java Virtual machine) is at the Processing nodes for the execution to take place. This would also require a descriptive way to tell the underlying Grid infrastructure how to execute the job.
3. It could be that the application already exists on the remote Grid network and all you need is a way to run the application after successful logon to the Grid network.
4. Another scenario could be that the application is transferred to a Vendor store and when the Job is issued the transfer takes place from that store to the Grid environment. In cases like this, the Vendor Store may have a better communication medium to the Grid environment than the user location.
5. A scenario can be to have a Proxy server to the Grid environment. The Proxy server acts as a gateway between the application and Grid environment and resides at the application local network but gives full access to the Grid environment.
6. It can also be that the user application is on the same platform as the underlying Grid environment and the user has a front end to the Grid environment in which case no further action is needed for its execution.

When a user decides to run his executable, after all the initial process of determining his identity by logging on to the Grid, the next major step is how to run his program. Figure 3.2 gives an overview of the proposed steps that the Grid user interface would take for a successful remote execution of a user's executable.

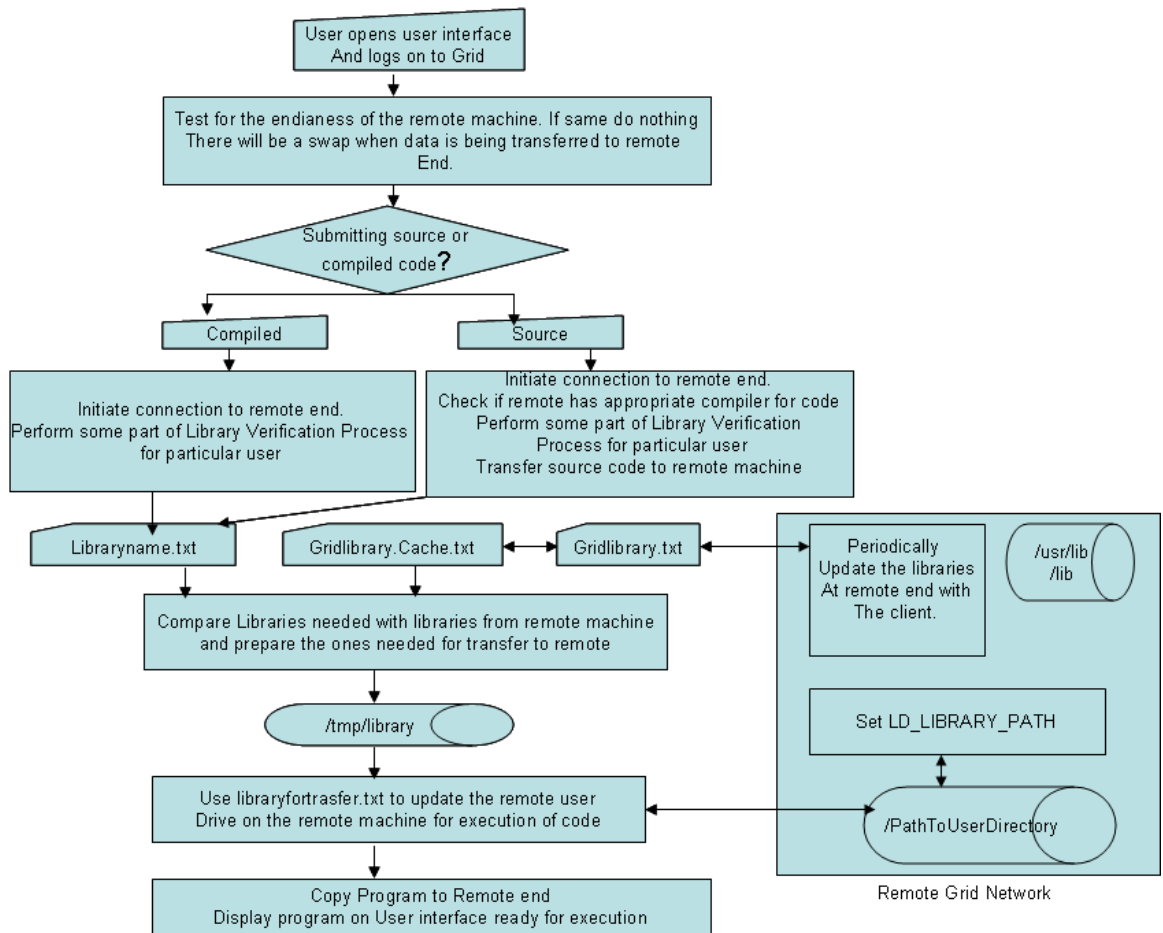


Fig 3.2: Flowchart of Job Submission Process

3.7.2.6 The Library verification process

This process is a sub-process in the overall job execution process. It ensures that a successful job will run in the remote end. In the research we need to find a way of keeping a database of the libraries existing in the remote machine, their versions and how recently they have been updated. Presently there are no central library servers to perform a comparison analysis between remote libraries and local libraries at the users end. We also need to find a way of transferring particular libraries needed by a user's code to the remote in a safe and efficient manner.

When running an executable, the dynamic linker loader lib.so or ld-linux.so loads the shared libraries needed by a program, prepares it to run, and then runs it [99]. If the static option is not used during compilation; all Linux programs are incomplete and require further linking at run time. The necessary libraries needed by the program are searched for in the following order

1. The environment variable `LD_LIBRARY_PATH`
2. From the cache file `/etc/ld.so.cache` (contains a compiled list of candidate libraries previously found in the augmented library path).
3. In the default path `/usr/lib` and then `/lib` or sometimes `/usr/local/lib`. The `/usr/local/lib` directory is sometimes not included in the `/etc/ld.so` file for some Red Hat Linux distribution.

As explained in previous sections, a remote machine would only allow the root user in its local domain the right to copy libraries to the `/usr/lib` and `/lib` directory. We are then left with the option of using the environment variable `LD_LIBRARY_PATH`. We check existing libraries in remote end with the ones on user machine to see which ones need to be transferred. We then transfer the libraries to a remote directory that the user has access to and then we update the `LD_LIBRARY_PATH`. A Standard script is used to achieve this. An example script (**updateLdLibraryPath.sh**) is shown in listing 3.3

```
# !/bin/sh
LD_LIBRARY_PATH=/pathToUserRemoteDirectory:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

Listing 3.3: Updating the `LD_LIBRARY_PATH`.

We then send this script to run remotely to set the library path and then run the executable. We update the cache library at the client end to keep a record of libraries used by previous programs for reuse by later ones. The problem with this approach is we get multiple copies of the libraries in the remote end but this would ensure that the user cannot tamper with directories that are important.

Library Verification Procedure

This process is part of the User Interface Program designed for users to submit jobs and would work for either a source or compiled code submission (illustrated as a flowchart in figure 3.3).

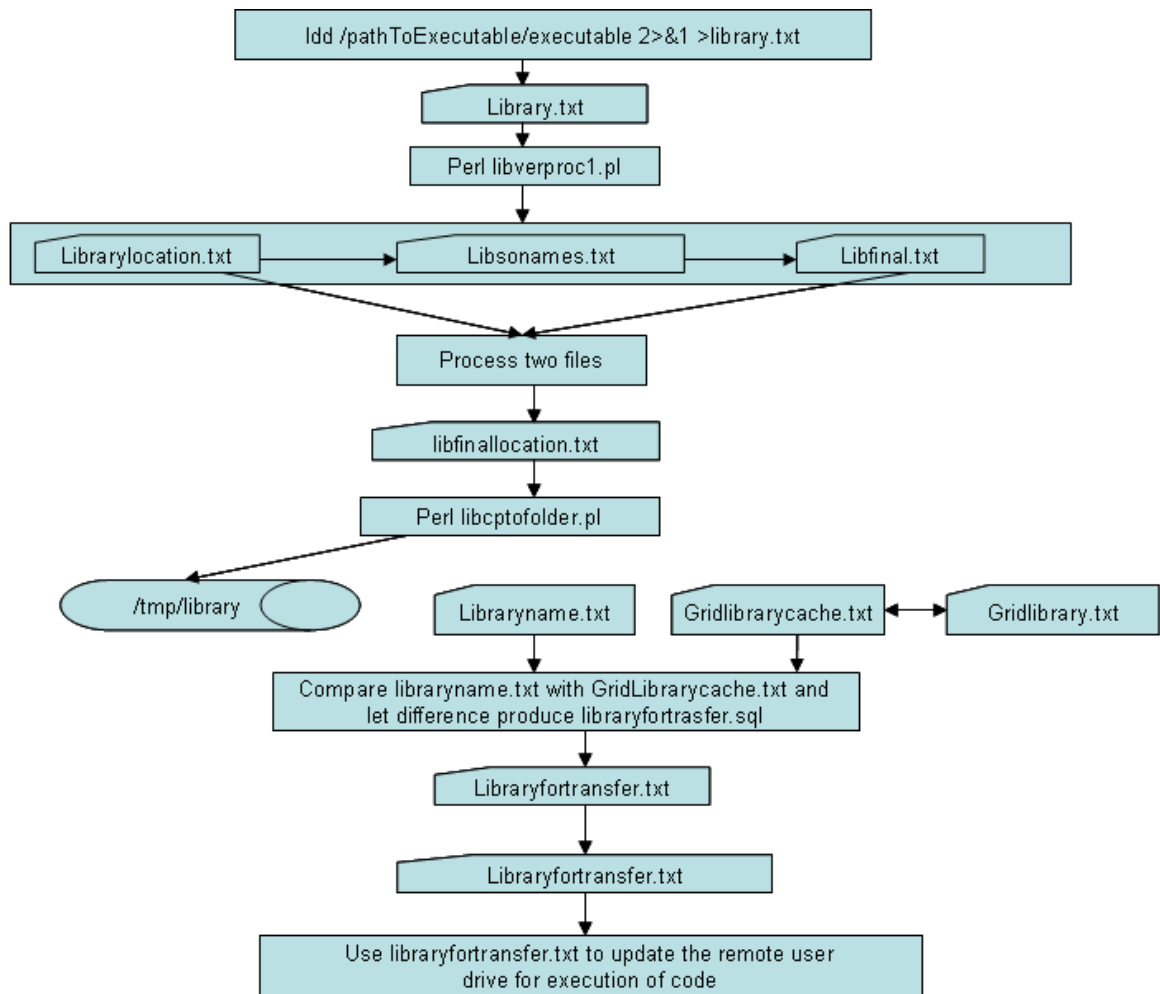


Fig 3.3: Flowchart of library verification process.

The standard Library Verification Procedure using an example code called *executable* is as follows.

- The name of the compiled code e.g. *executable* is given by the user.
- Run the following command `ldd executable` on the code.
- The output is captured and stored in a file *library.txt*.
- Run perl code (*plibcptofolder.pl*) that goes through *library.txt* and performs a series of text processing, producing various files until a final file (*libraryfinalalloc.txt*) which keeps information about the exact library files is produced.
- This file is then processed to get a file *libraryname.txt* which is then compared with the library already at the remote end using a perl program that goes through the two files and find the strings present in local library but not in the remote library.

- The files are compared until the libraries not present in the remote library file are captured. These files are then transferred to the remote end before the source code is compiled for execution or the compiled code executed.

Figure 3.3 shows the flowchart of this process and figure 3.4 gives a detailed flowchart of the library verification process. The process is detailed in the following steps:

1. *ldd agah* produced the following output stored in a file called *library.txt*

```
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0xb7ef3000)
libm.so.6 => /lib/tls/i686/cmov/libm.so.6 (0xb7ecd000)
libgcc_s.so.1 => /lib/libgcc_s.so.1 (0xb7ec2000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7d8d000)
/lib/ld-linux.so.2 (0xb7fe4000)
```

The highlighted lines are the libraries that the executable depends on.

2. A perl program *perl libverprocl.pl* goes through *library.txt* and captures the absolute path name of each of the *so* file and saves it to a file *libraryloc.txt*

```
/usr/lib/libstdc++.so.6
/lib/tls/i686/cmov/libm.so.6
/lib/libgcc_s.so.1
/lib/tls/i686/cmov/libc.so.6
```

3. The *so* files are actually symbolic links to the real library files and so the perl program goes through each file and captures the exact library files that the symbolic link point to. Some don't point to any file. This produces the following result and this is stored in *libsonames.txt*

```
lrwxrwxrwx 1 root root 18 2007-03-07 15:30
/usr/lib/libstdc++.so.6 -> libstdc++.so.6.0.8
lrwxrwxrwx 1 root root 11 2007-03-07 15:41
/lib/tls/i686/cmov/libm.so.6 -> libm-2.4.so
-rw-r--r-- 1 root root 40208 2006-10-08 19:21
/lib/libgcc_s.so.1
lrwxrwxrwx 1 root root 11 2007-03-07 15:41
/lib/tls/i686/cmov/libc.so.6 -> libc-2.4.so
```

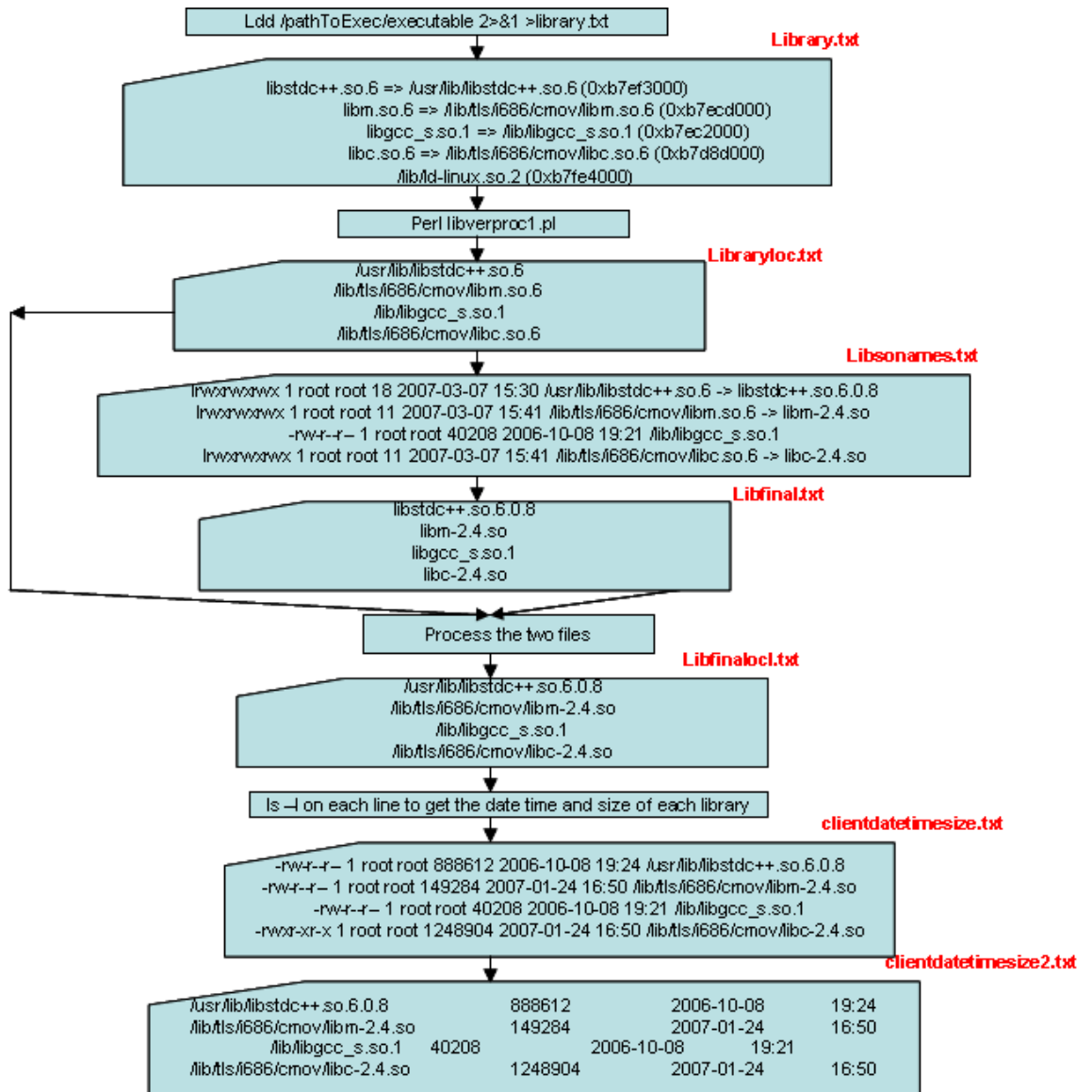


Fig 3.4: Detailed flowchart of library verification process

- A perl program then goes through this files and captures the exact library files needed for the executables which produces the output below stored in the file called *libfinal.txt*

```

libstdc++.so.6.0.8
libm-2.4.so
/lib/libgcc_s.so.1
libc-2.4.so
  
```

- The Grid user interface then takes the two files *libraryloc.txt* and *libfinal.txt* and produces a final file (*libraryfinalloc.txt*) which has the exact location of the actual libraries.

```

/usr/lib/libstdc++.so.6.0.8
/lib/tls/i686/cmov/libm-2.4.so
  
```

```
/lib/libgcc_s.so.1  
/lib/tls/i686/cmov/libc-2.4.so
```

6. The Interface then calls a perl program *perl libcptofolder.pl* that iterates from the beginning of the file to the end and copies the libraries to a temporary location in this case called */home/agah/tmp*.

There is a flat database file that consists of all the library files at the remote end (in directory */usr/lib* and */lib*). We copy all the library files at remote end to */tmp/librarymirror*. We run *ldd* on the libraries to capture their exact location and the date the library was created and the size. We then create a flat database with the *nameoflibrary*, *datecreated*, *timecreated* and *size*.

7. We call this file *gridlibrary.txt* and we keep a cache copy *gridlibrarycache.txt* which we will always compare with the libraries needed by the users' application stored in *clientdatetimesize.txt*. We already have the size of each of the library. We then use *cksum* to compare this size with the size of the same libraries in *gridlibrarycache.txt* to see the available bytes in the file. If the *cksum* (the program that gives the size of a file) of a file is the same and the date created is the same then we can assume that the libraries are the same. As stated in the previous paragraph, this particular library verification process is based on the assumption that our method is sure but we can never be certain that the versions are entirely the same as this would work for a specific scenario and may not work generally. If the libraries are not the same, we transfer the changed library to the remote end before user attempts to run his job. All library files start with *lib*. Most remote machines will not allow you to copy libraries from the client end to their default library path (*/usr/lib* and */lib*) but we still need to keep a database of existing libraries at the remote for the comparison purposes.

The library at remote end is called *gridlibrary.txt* and to obtain this, we run the command, *ls -l* on */usr/lib* and */lib* to capture all of the libraries. Careful study at the way libraries are stored on the x11 operating system has shown that it will be hard to keep updated copies of the new libraries. It is beyond the scope of this project to decide how best libraries can be stored. But assuming that all libraries are stored properly in the */usr/lib* directory then we can always check

it periodically to update the flat database and then compare with the libraries at the client end before any transfer occurs.

When we tried to see the output of the `/usr/lib` we discovered that some *sonames* (symbolic link to library real name) referred to libraries in a different location. The standard procedure would be once a user's library is checked against the library at the remote end, if the library he wishes to use is more updated then it is transferred to the users remote end directory and the `LD_LIBRARY_PATH` set to make the executable know where to pick the dependent libraries.

8. The interface then calls a process which will initiate the command *globus run* to copy each of these libraries to a designated location at the remote end waiting for execution.
9. The user can then initiate execution of his job from the Grid user interface.

3.7.2.7 Running sample executable from remote end.

As explained in the previous section, the process that takes place when a user tries to submit a source code or compiled code is described in the above sections. We now illustrate experiences of executing a sample source file *agah.cpp* and compiled file *agah* from the client end to the remote end.

When we tried to compile a source code *agah.cpp* at the remote end, copying the files to user end was quite straight forward. Compiling gave errors that were particular to how compilers understand source codes. A `c++` source code that was edited on windows with visual studio flagged an error when we tried to compile at the Linux remote end. It complained of line endings. Further investigation showed that users must use editors in windows that were acceptable in Linux. For a code developed on a Linux machine, compiling at the remote Linux machine went well. Errors that can occur here would be syntax or semantic errors from the program itself. The Grid user interface is able to capture the errors from the *Stdout* and *Stderr* and this is displayed in the Status bar of the Grid interface. The users can then check their code, make necessary corrections and recompile again. We were able to compile the executable,

which once successfully compiled, displayed on the remote end and we were able to drag the input file that produced the final output.

We also tried to run a code compiled code at client end, on the remote end. For a windows user, this is not possible as a compiled code in windows would not run in Linux. We then tested with the code compiled in Linux. We went through all the procedures of determining the dependent library files, copying these files to remote end and then trying to run the code, we got ‘segmentation fault ‘error which meant the program was trying to access a part of the operating system not allowed. Some of the problems actually come from the way libraries are stored in Linux and the kind of protective system Linux runs to shield the operating system from external programs attack. In the earlier part of the project we tried to compile codes on windows machine, ran *depends* software (gives the dependent library on a compiled code), on the compiled code and copied those libraries to another windows machine and was able to run it successfully.

We have seen that using the simplest of code does pose problems when trying to recompile on another machine. We have been able to determine that a universal editor is useful so that recompiling across platform would not give problems of line editing. It has also been determined that due to the way the Linux operating system works, recompiling codes on a different machine is a daunting task. We also showed that running a compiled code on windows could be simpler but most Grid environments don’t usually have windows operating system as part of their processing nodes.

3.7.3 Data and Application

This section discusses users concern about where their data or application should reside when using the Grid. This concern is investigated below in the way of looking at the three different scenarios that can occur. A scenario was used in this section to look at the way the Grid user interface could be presented to a user and what concerns the user would have in using this environment. A windows interface is created to allow a user submit a job to the Grid that involves the user’s data, and application that can act on this data. There are two physical locations: The client and the Service Provider. The Service and data could be in any of these locations in different ways

and each scenario is itemised and looked at to see the feasibility of implementing the windows interface.

Scenario 1. Data and Application resident on the Service Provider

Issues:

1. This is the simplest of the entire scenario and the best way to implement is via a portal and it is the job of the service provider to provide the user with all the capabilities offered by the service.

Scenario 2. Data resident on the Service Provider and Application on Client

Issues:

The same as above but in this case the main concern is the data leaving the provider.

1. The Application provider may not be confident to release data because of security and trust issues.
2. If the trust issue is not a problem then the next issue may be how large the data is. Because the size may determine whether it is worth the transfer.

Scenario 3. Data resident on Client and Application on the Service Provider

Issues:

1. The Application provider may not be confident to release algorithm because of security and trust issues.
2. If the trust issue is not a problem, the size of the application may be large and this would determine whether is worth the transfer.
3. Transfer of Application to the remote location (Bandwidth Speed).
4. Discovery of the best service that can work on the data. (Services have predefined data formats that they can work on.) Of course if that service is via Globus the OGSA-Data Access and Integration (OGSA-DAI) [82] layer can handle the various data formats transferred across different platforms and locations.
5. Knowing the format the service requires. Services would be stored in a standard way, describing what they do and the type of data they accept. The WSDL standard would help in how services are described and discovered

For all these scenarios the one thing common to all is the issue of where the data would be stored before any processing can be done on it. There are data storage providers available but the current implementation is in form of a virtual private network, which is far from what a Grid environment should look at.

There are many ways of presenting Grid applications to users. The report by Dr Tom Harris (Harris T, 2004) proved that some Organisations are highly concerned about the amount of Data they regenerate per transaction and how to handle this data. One way of solving this problem could be to have a Data Storage Provider in the Grid where all the organisations data are stored. All you may need to do is create a viewer to that application from the client end while the Data is resident in a remote location. Getting customers to agree to this would also depend on a lot of trust and security issues, as they have virtually no control over their data. It is therefore necessary to say here that because this technology is evolving, there are a lot of concerns from users. But as more people embrace use it and the standards become well known and stable a lot of these issues would not be a problem.

3.8 Thesis Constraints

This project has been constrained by a lot of factors and this has contributed in determining the approach that has been taken. Grid computing has come with a lot of challenges discussed earlier and as such requires a lot of time and effort in investigating all the possibilities that it offers. Therefore we are only able to focus our research on some specific kind of users and a particular type of approach in building grids. The particular approach chosen has various ways in which it could be implemented and this in itself would require a great deal of understanding of the Grid technology.

As discussed earlier the capabilities offered by the interface was demonstrated with the use of a sequential code and one processor. Scientific applications developers would normally have parallel codes but the interface is built as a platform for future work on any type of legacy codes. We believe that we have been able to demonstrate the usability benefits of the new user interface to the Grid. We also have been able to

bring attention to the importance of libraries used by the legacy codes and how they can be accommodated by the underlying Grid environment.

3.9 Conclusion

This project is very particular about presenting to the user an interface that is simple and usable by people with limited computer knowledge. We present to the user, an interface that is nice and intuitive. Users requirements were captured from the interviews conducted with them. The ways users currently use the Grid was considered and the proposed way of using the Grid was presented. The various issues that involves Grid-enabling an application were explored and the tools for implementing the interface have also been explored. Users' tasks have also been analysed and we are also able to present what an average user would expect from the Grid on their daily use of the Grid interface presented to them.

The current system implementations are designed in a way that the user has to learn a new programming paradigm to be able to use the Grid effectively. This project aims to deal with these very complex issues by doing a thorough analysis of the common programs users work with and how the Grid can accommodate these programming codes. This has made us to focus a lot on how libraries work, how to move them to the remote end, how they can be updated and how they can be verified for use with different users and different jobs that are executed on the underlying Grid environment. We realised during the library verification process that it is very difficult to be sure which version of libraries exist on the remote and client end during comparison. I have developed a way that I feel would help to determine that the versions match or not. But this is not 100% effective and reliable and as such I feel that the operating systems must find a very reliable and neat way of updating different versions of library to make it easy for the Grid-enabling process.

4 Graphical Grid User Interface

This section involves the overall process that was involved in implementing the interactive interface for users to submit jobs to the Grid. As explained in previous chapters, the two main scenarios for users of this software are for a submission of a source code for compilation or the execution of an already compiled code. The Library verification process was also embedded in this interface and standard conventions and text were used in designing the interface to make it easily accessible to the user.

In the implementation of this interface, series of processes were involved. We looked at a high level process that is involved in a user's job submission to the Grid. We then analysed the user's day to day task, then proceeded to the design and implementation of the interface.

The Graphical Grid interface has been implemented using the QT software using the C++ programming language. Embedded in it is the COG toolkit (a client software for the submission of job in a Grid environment using Globus toolkit). With the user's requirements established, a prototype was designed showing the four windows proposed for the interface and the proposed functions that will be implemented in these windows. The next step was then to setup a workbench simulating the client machines and remote machine. We then started writing the software for designing the interface whilst embedding the underlying functionality. The interface developed is semi-functional and was able to demonstrate a scenario where a user submitted a source code to the remote Grid network.

4.1 Tools for Implementing the Grid User Interface

The interface was built using different software's. The white rose Grid was the intended remote end for the underlying Grid environment and the Grid middleware used is the Globus Toolkit. We could not develop and test the software implementation process on the white Rose Grid (WRG) because the WRG could not take the risk of any of the machines being corrupted. To improvise, we had three

virtual machines representing a linux client, a windows client and a linux remote end to represent the underlying grid remote network. We then installed client software (COG Toolkit) to allow us access the remote Globus toolkit. This toolkit is responsible for storing user's credentials at the client end for authentication to the remote end. It is also used for submitting jobs and querying the underlying Grid remote end. This has to be properly set on the client end. The QT programming tool implemented in C++ was used to build the Grid interface due to the fact that C++ is statically typed, efficient, portable and avoids features that are platform specific or not general purpose. The Cog kit is embedded in this interface.

4.2 User Centred Design process

The Design approach is user-centred, as a good user interface design is very critical to the success of a system. Computer users now expect applications system to have some form of Graphical user interface (GUI), which supports high-resolution colour display as well as interaction with a mouse and keyboard. The advantages of GUIs are (Sommerville, 2001):

- They are relatively easy to learn and use. Users with no computing experience can learn to use the interface after a brief training section.
- The user has multiple screens (windows) for system interaction. Switching from one task to another is possible without losing sight of information generated during the task.

The design process after the requirements was established is discussed in the section 4.3 which presents the user interface prototype with reference to the windows (showing the functions and requirements considered) and architecture of the interface. Section 4.4 uses the concept of the Model-View-Controller [95] to show the key flow of information between the data store and the user interface. Section 4.4 shows how the environment was set up. Section 4.5 explains the classes of the interface with a class diagram showing their relationship and Section 4.6 presents the screenshots [Appendix D] of the user's interaction with the interface

4.3 User Interface Window components

This section discusses the user interface main components which consisted of the four windows and the functions and requirements that were needed for each window.

4.3.1 My Workspace Window

This window aims to present to the user any output from the jobs they run. The window also aims to allow the user have access to all their compiled codes ready for execution.

Requirements:

Connection to a data repository where results can be viewed by user

Functions:

- Ability to show new results.

4.3.2 Services/Applications Window

This particular window allows a user to drag the instance of their compiled code or other services that they would like to use on the Grid. Users are able to drag their input file needed by their compiled code to initiate the job process.

Requirements

- Connection to a server where compiled codes/services can be deployed.
- Connection to Grid services.
- Method required for interrogating Grid services.
- System to start interrogation for example a button to periodically update available services.

Functions

- Get a description of Grid services from the registry (WSDL).
- Accept drag and dropped service icon which deploys an instance of the compiled code/service on that server/Grid.
- Ability to run the service and a way to start it for example, a run button.
- Ability to accept data sources dragged in.
- Ability to link data to a Grid service.

- Represent the services/compiled code graphically.

4.3.3 Available Grids Window

This window aims to display Grid environments available to the User as well as Users recently compiled code that can be run. All successfully compiled codes/compiled source codes are displayed here for the users.

Requirements

Method to query repository for finding Grid environments using for example MDS-GRIS/GIIS.

Functions

- Creates an icon for each Grid found from the Repository of services.
- Creates a new pane when a new service/compiled code is dragged to “Services/Applications window.
- Allow dragging of description icons into “Services/Applications” window.

4.3.4 Data Repository Window

This window displays all the data sources both locally and remotely to the user. Users are able move data to and from this data source using drag and drop.

Requirements

Method to connect to arbitrary sources (for example a storage resource broker, conqueror, Gnome etc).

Method to add new data sources with useful names to make it clear to users.

Functions

Allow a data source to be dragged to “Services/Applications” window where it can be used as a source for service/compiled code.

Allow results to be pulled back to a user’s local space if desired by user.

Ability to set preferences for users’ local and remote data.

4.4 Interface Diagram

The interface diagram models the interaction that the users have when they use the software for a successful execution of a job. This was carefully designed before the programming of the interface commenced. Figure 4.1 depicts a particular way in which a user can initiate his application/compiled code to be submitted onto the Grid.

It shows the overall high level process of logging onto the Grid, submitting a source code for compilation/compiled code, and then dragging his input data on that code to trigger successful execution. The job is completed and the result returned to the user's workspace.

This diagram has been able to illustrate the various components of the Grid user interface and how these components interact with each other to bring about the successful implementation of an improved usability of a Grid-enabled environment.

The diagram has been able to illustrate the following:

- Users can log on to the Grid in a simple manner.
- Users can describe their jobs in a simple and effective manner.
- The jobs are thoroughly checked for verification by going through the library verification process.
- Users are able to interact with the system easily with drag and drop.

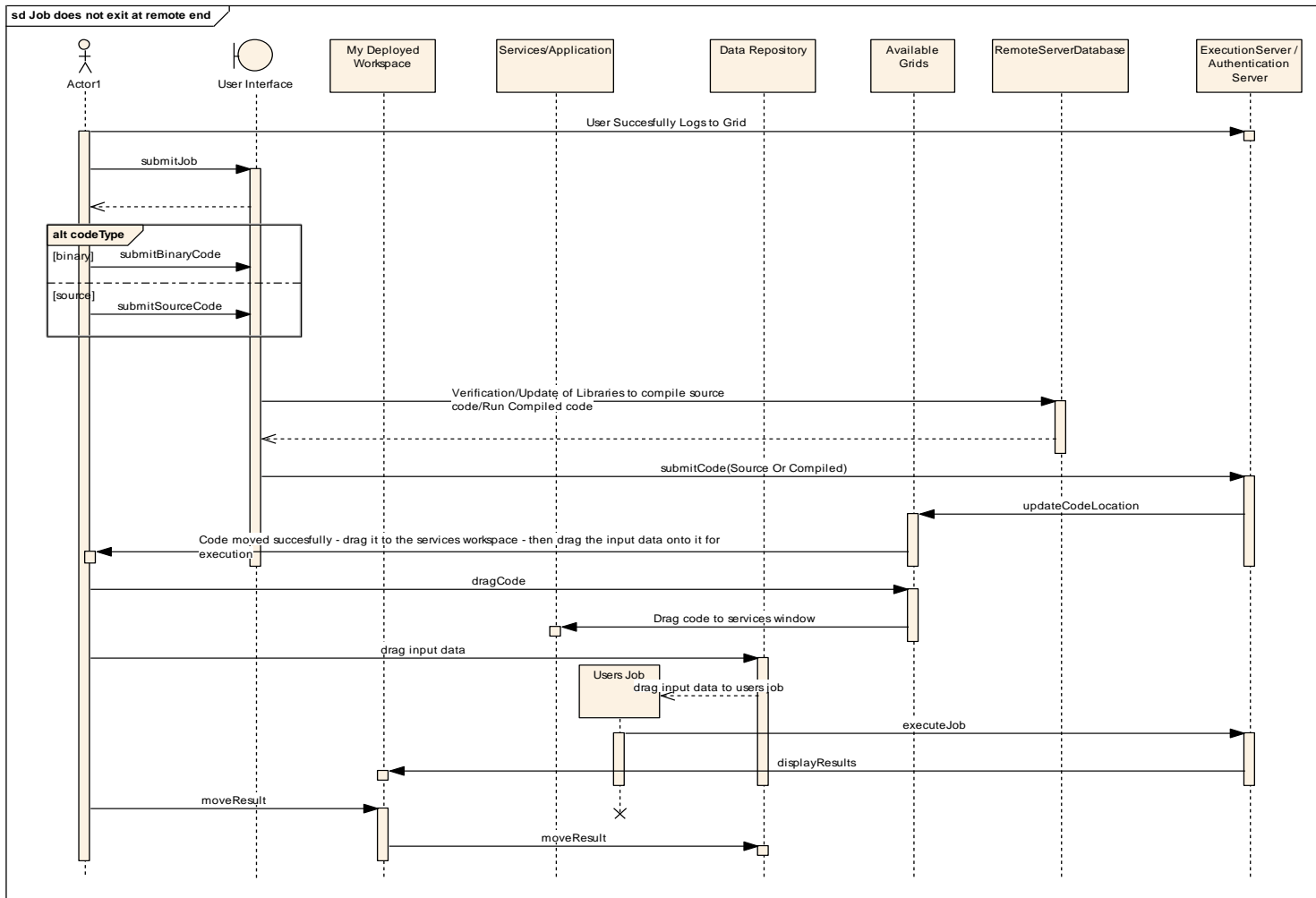


Fig 4.1: The interface model.

4.5 Setting up the environment

The interface was intended to be designed for both Linux and Windows users. We needed three systems: The Windows Client user, The Linux Client and the Linux Server. Each of the Windows Client and Linux clients had the following installation:

- Visual Studio for windows and gcc preinstalled for Linux.
- A cog kit installed and the environment set properly so we can issue the 'globus' command from the command line without the full install path.
- The Grid Proxy certificate installed.
- QT installed and integrated with the *visual studio* and *gcc*

The following steps were taken in the successful setup of the environment on which the Grid interface was built and tested. The overall process (Appendix C) is summarised as follows. The initial setup involved setting two computers to act as the client and server respectively. Initial development platform was windows but we moved to linux when we started the library verification process due to some constraints in the windows platform. The final setup used the virtual machine option where three computers were created with two acting as client and the other as a server. This option allowed development to be flexible as the virtual machines could be accessed from anywhere regardless of location

The next stage involved setting up the remote virtual machine which acted as a server for users to submit their jobs. Installing and setup of the remote linux virtual machine (*agah-desktop*) was very challenging and cumbersome. The virtual machine was created with the necessary memory for the harddisk, RAM etc. Then we installed Globus on the machine. Globus installation was very time consuming and challenging and the most important aspects involved the following steps:

- Installation of Globus, the Grid middleware that orchestrates user authentication, job submission, monitoring, resource allocation, and data management.
- Setup of the Certificate authority (CA) to allow users login remotely to submit jobs.

- Requesting the Host certificate on node *agah-desktop* to make sure that the previous step was successfully setup and signing the certificate using the CA setup in the previous step.
- Setup Remote Drive. For the test bed demonstration we used the smbclient service to access the remote drive of the Grid network. Smbserver service was installed on the Server end so that users are able to mount a remote drive from their client machines. We were also able to map the same remote directory on the windows client. Figure 4.2 shows the workbench setup of the software development.

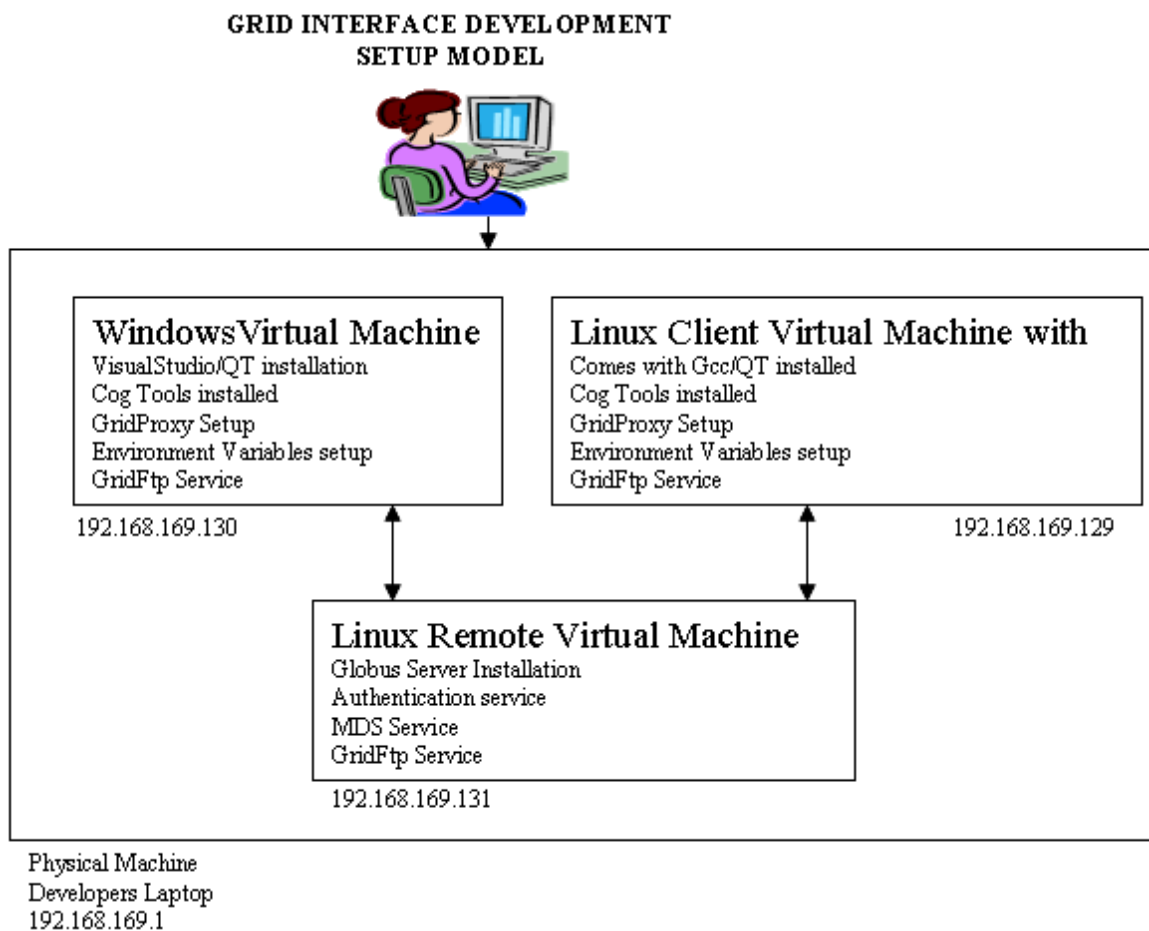


Fig 4.2: The Workbench of the Grid User Interface

The next step was the setup of the client virtual machines. As explained above, there were two client machines during the programming of the interface: one installed with windows and the other with linux. The following steps ensured proper implementation of the Grid user interface:

- Installing QT/C++ Compiler

- Install and setup of Globus using the COG kit and making sure that a test user can successfully log on to the remote end.

The final step in the Grid user interface was to setup the interface to submit job on the white Rose Grid (WRG). As mentioned earlier, the programming development took place using the virtual machine setup. When the interface was developed to some extent, it was necessary to explore how user's setup accounts on the WRG. This process is explained in full in Appendix C.

4.6 The different classes of the interface

In designing the interface, five major classes were created. The main window class. The Services Class, The Application class, The MyWorkspace class and the DataRepository class.

4.6.1 QMainWindow Class

This is the main class in which the interface is designed. From this class all the objects that are needed for the interface is created and initialized. The design is such that four main windows are created. They are represented by four classes: MyWorkspace, Application, DataRepository and Services. All the classes and sub classes all have a reference to this class to enable them access other private objects.

4.6.2 MyWorkspace Class

This class is responsible for displaying the results of a users completed job. The user can then either copy the result to the remote directory by dragging it to the data repository or leaving it in the local store.

4.6.3 Application Class

This class displays the codes that have been compiled at the remote location. It also displays already compiled codes that are transferred from the user's local machine.

4.6.4 Data Repository Class

This class represents the Data Repository of the interface and the underlying Grid interface. At the moment this window is represented by mapping virtual drives to the remote Grid network and also displaying the user's local directory. Users are able to drag and drop files to this window.

4.6.5 Services Class

The services class allows a user to drag his compiled code to a button. The user can then drag the input file/data that will trigger the execution of the code. All these have subclasses and a UML representation of these classes is presented below.

4.6.6 The Class Diagram of the interface.

The different classes of this interface have been described above and how each one aims to achieve the functional and usability requirements of the system. These classes are a representation of the way the Grid user interface has been programmed. There are other classes created which is either a subclass of the window or of the main program. The class diagram aims to present a summary of the entire code written for a particular software and how each class interacts with the other. This interface is presented in figure 4.3

4.7 The Grid User Interface Software Setup

From all the previous sections explained above, we can draw up a kind of setup instructions for any user that wants to use the tool for submission of Job to the Grid.

Interface Requirements

1. User uses a universal editor that would not flag error in a different programming environment.
2. The interface supports currently C++, C Languages.
3. User must have a valid eScience certificate.

Setting up on Users Machine

1. Install Cog on user's machine on C Drive and set the environment variable properly.
2. Setup user's globus certificate properly on users machine and test the cog-proxy-init.

3. Map user's remote home directory on the N: drive.
4. Install the interface on the C:\Gridinterface directory.

4.8 Job submission Process

The job submission process for a user is depicted as a flow process and explained in the following steps.

1. The Grid user interface is launched.
2. User logs to Grid and goes to step 4
3. If User does not log on to Grid then go to step 2
4. Do you want to submit a source code or compiled code?
5. If user wants to submit source code go to step 7
6. If user wants to submit compiled code go to step 8
7. Interface asks user for source code, ask user for libraries used by the source code and input file needed by source code, user then hits button to submit
 - a.) Interface will take the library listed by user and check with remote database if they exist and only transfer the one not existing
 - b.) The source code is then compiled and if any errors, reported to the user.
 - c.) If not errors the interface then goes to step 9
8. Interface asks for the compiled code, gives user a button to list all the dependent libraries and ask user for the input file and user hits button to submit
 - a.) Interface will take the libraries listed by the ldd tool not seen by user and do a thorough library verification process to check the correct libraries needed by code, exist at remote end.
 - b.) The libraries not in existence are transferred and when this is done successfully, the interface goes to step 9
9. The compiled source code or already compiled code is displayed for user in the application section of the interface.
10. The interface then prompts user to drag the input file onto the compiled application.
11. The result is displayed onto the user's workspace window.

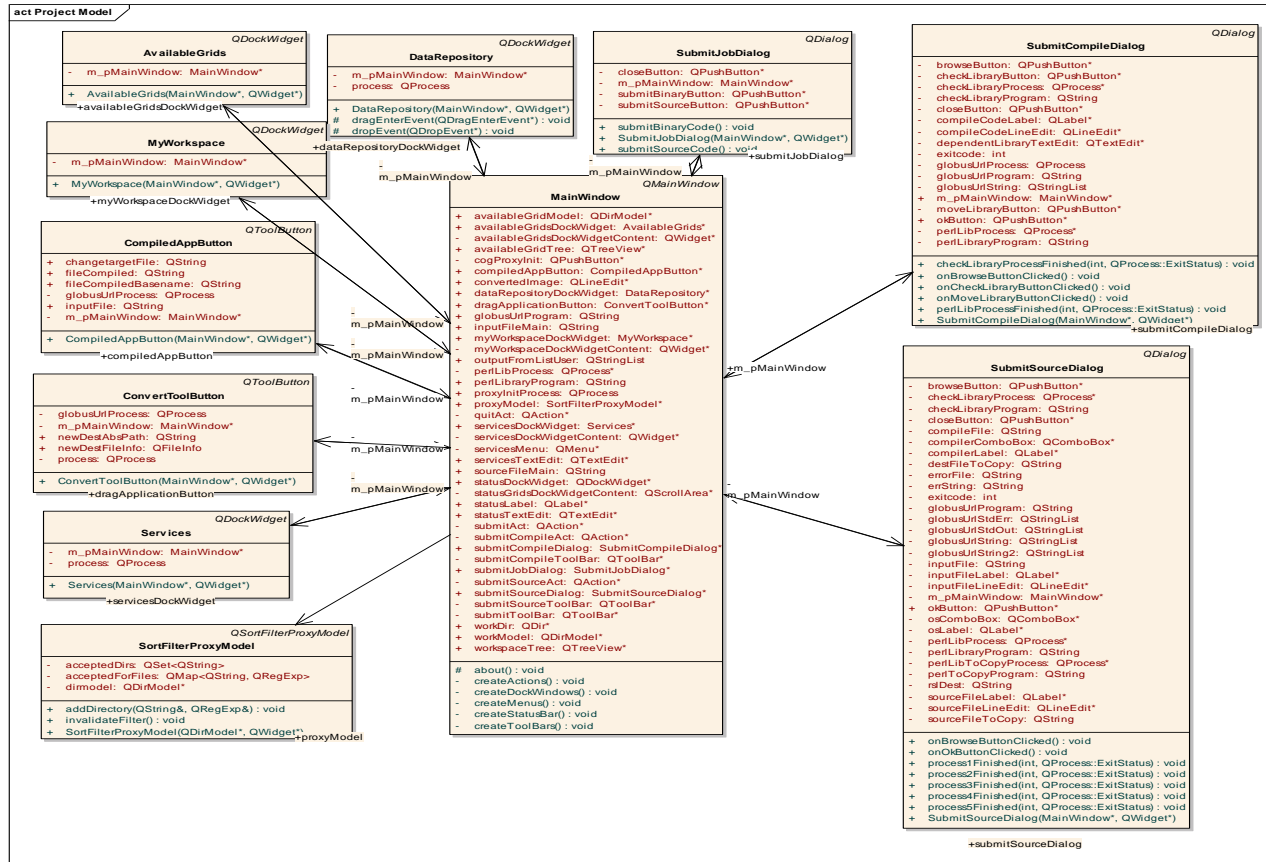


Fig 4.3: The class diagram.

4.9 Conclusion

The overall process of building the interface was thoroughly investigated by looking at a high level process of what it entails for an application to be Grid-enabled. Looking at this helped us to decide the realistic features that would be embedded in the Grid user interface to guarantee that the user's requirements will be met.

The Grid interface software is a semi functional interface. It is primarily designed to show how we can build interfaces that make Grid-enabling applications easier and less cumbersome for potential Grid users. The functionality of the interface was well explored and highlighted in this section. The interface explored the possibility of submitting a source code or a compiled code to the Grid and looked at the procedures and steps that have to be followed for this to be achieved.

The source code submission was explored using simple C++ code as a test code. But this interface is built in way that allows for further development for other programming languages. A snapshot of how a user can submit a C++ source code is shown in appendix D.

5 Evaluation

Usability inspections methods demand that when an interactive interface is built there must be some way of measuring the effectiveness and usefulness of the product. This particular interface is a semi-functional prototype of submitting a job in a grid-enabled environment. The interface aims to prove that there are easier ways of grid-enabling an application and the steps taken to do this can be more effective. In the evaluation process the original users that were interviewed were not in the final evaluation due to their absence at the point in time. This would not invalidate the evaluation results as a careful selection was taken from a set of Grid application developers (Some of who are actual Grid programmers and others are involved in eliciting usability assessment of Grid applications developed). These users can be a representative of the original Grid users as they themselves work with the Grid and able to critic an interface that they have not be involved with (folstad, 2007).

The overall objective of this project is to reduce the steps taken to Grid-enable an application and to design a tool that allow users submit jobs more effectively than before. From these objectives, we reviewed past Grid projects and later interviewed actual users of the Grid. We then came up with the following system requirements:

- Users want to submit their existing code via interface without modification.
- The user wants to have their Job run successfully.
- The users want to recover from errors easily.
- User wants useful feedback and Monitoring throughout interaction.
- Users want icons that depict the actions to be performed.
- The interface will allow users perform drag and drop.
- User wants some Grid ‘commands’ represented with Buttons on the interface.

We have conducted an evaluation with these set of users to see if these system requirements have been met. Based on the evaluation conducted which is discussed in the next sections, we have been able to discuss the requirements that were met. The interface is simply designed and users can interact with it in a straightforward manner. The interface is also designed in a way to allow users transfer their source codes for compilation and can either give useful feedback on errors encountered or a successful compilation ready for execution. Users are also able to successfully execute their

codes/application and have their result displayed for them. The steps that a user goes through to achieve these requirements have also been minimised so users don't get frustrated in their quest to use the Grid.

5.1 Conducting the evaluation

The evaluation of the interface was conducted by a test group consisting of six users of the ACAG group and a usability expert. These six users are expert programmers of the Grid and expert computer Scientists. They were chosen for the evaluation as they have been involved in different projects with the Grid and would see the interface from the user's point of view. The domain experts consisted of two groups. One Group consisting of three users performed a study of the system with the designer present and answered questionnaires after the study. The other group consisting of three users performed a study of the system with the designer present and a usability expert recording the events. The usability experts also performed a heuristic evaluation. These three separate analyses are reported in the next section.

The Grid user interface was analysed by using a specific task of submitting a source code to the remote network, compiling it and eventually executing it remotely. This procedure was given to each of the users and when users finished the task they were given questionnaires to answer based on the task they had just performed. This instruction is found in Appendix E.

5.1.1 Use Case for Evaluation

Use case is a part of software engineering that helps to describe the way a system behaves from a user's point of view. It is a technique used to capture a system's behavioural requirements by detailing scenario driving threads through the functional requirement [103]. In this particular Grid user interface, the typical task a user would want to perform is to either submit a source code to the Grid for compilation and then running it or running an already compiled code. The use case here is for the source code submission process and this is given below:

Use Case for Job Submission to the Grid Explorer

1. The User logs on to the Grid with password
2. User clicks the submit source code button and enters the source code, input file needed for the job, the operating system and the compiler.
3. User waits for the job to be compiled and it appears on the interface
4. User drags the application to the services window ready for execution.
5. User then drags the input file to the application to trigger execution
6. User then waits for result which appears on the user workspace.

The evaluation was conducted with this use case. Most especially the evaluation in section 5.1.2 and 5.1.3 was conducted using this use case plus explicit instructions on how to interact with the interface (Appendix E). When we conducted the evaluation in section 5.1.4, we used only this use case to see whether users can interact with the system without any specific information.

5.1.2 Results from three domain experts (Questionnaire and think aloud)

The Grid user interface was presented to these three users with a guideline (Appendix E). They followed these instructions and gave their feedback using the questionnaire (Appendix F). Think aloud technique was used in conjunction with the questionnaire as sometimes questionnaires can be less probing so it was intended that when users share their thoughts as they go through the task, we could find more user preferences and requirements that the questionnaire could have missed. The analysis of the questionnaire and feedback from think aloud is given below.

The evaluation technique used here was observational and query (Think aloud and Questionnaire). The questionnaire had nine questions and each one had the option of 1-5 where 1 represented strongly disagree and 5 strongly agree. The questions were ranked and open-ended and evaluators could rank each question and make comments on them if they wanted. Table 5.1 presents the response from each user.

Questions	User1	User2	User3
The Start-up user interface depicts what the system wants to achieve.	No Comment	The interface could be a little clearer like change the Available Grids window to Available Grids/Executable and Data Repository window to input/Outputs.	I am aware of the development so I know roughly what it is designed to achieve. Remapping the panes so that the main flow of actions is top left to bottom right or changing to more of a wizard based system would be helpful to novices
It is easy to recover from mistakes.	At least can start a new task quickly	If I dropped the wrong executable on the run button, I would need a 'clear' button (or reset)	No failures occurred so cannot comment
It is easy to get help when needed.	No comment	There are useful help messages but would benefit from more help	I was stepped through the process so cannot comment
I always know what the system is doing with appropriate feedback.	More prompts for operation results could be better	I wasn't sure what was going on at 'logon to Grid' stage.	It was not obvious why the system was asking me to 'create' something when logging on. I recognised the Java Cog kit proxy tool but a novice user might not.
I always know how well am doing.	No comment	Sometimes it was not clear how well I was doing – eg logon to Grid	Mostly it is the case but some buttons draws the focus to the top right but status messages are at the bottom. Having the drop buttons nearer the status messages would be useful or pop up dialogs (switchable) for prompts

Table 5.1: Users' response to questionnaire.

The system tells me what to do at every point.	No comment	Without the instructions, I would not have known what to do.	Mostly it is the case but some buttons draws the focus to the top right but status messages are at the bottom. Having the drop buttons nearer the status messages would be useful or pop up dialogs (switchable) for prompts.
The interface was easy to use.	No comment	Drag and drop is good but some parts are not intuitive.	Mostly easy to use but needs the panes moving around. The Grids pane has a file browser so should be renamed
The dialogs and menus are easy to understand and consistent throughout the interface.	No comment	Buttons are not consistent in shape appearance and changes when context changes.	The Cog kit uses a different look and feel which might be off putting. The dialogs are mostly explanatory apart from the focus issue. So some good points and some minor negative points.
The errors messages I see are clear and precise to me.	No comment	I didn't really see error messages.	There were no error messages

Table 5.1 continued: Users' response to questionnaire.

5.1.3 Results from the Usability Expert (Heuristic Evaluation)

This was conducted by a usability expert and although the minimum amount of expert you need for a heuristic evaluation is three this evaluation proved useful as it was possible to compare results with the other analysis and determine the major usability problems of the interface. As explained above the user followed the same guidelines and instruction and the heuristic evaluation result is presented in table 5.2.

Problem ID	Location	Description	Severity	Heuristic Violated
1	The Grid Explorer Main Window	Not clear which bits are my local machine and which bits are grid	Major	6
2	The Grid Explorer main window	First task is to log in – but not obvious	Major	1,6
3	Log in Window	Login dialog box remains open with password field filled in, don't know what to do	Major	5
4	Submit source code Dialog	Ditto input file – unclear whether this needs a file name or a file path	Major	6
5	Services/Application	“Drag your exec here” is styled like a button but it is a drag-target area .Not clear what to do with exec file	Major	4
6	Top Toolbar	Don't know what “submit” means because there are three different things to submit	Minor	2
7	Log in Window	Nothing on the screen that says I am logged in and log in button remains – confusing	Minor	1
8	My Workspace Window	Window to browse my workspace is tiny and cannot be expanded so it is difficult to manipulate and find files	Minor	7

Table 5.2: Heuristic Evaluation Sheet 1(Usability expert)

9	Submit source code Dialog	Submit source code dialog “type in source code” is a question but actually wants a file path	Minor	5,6
10	Status Bar	Status bar is used for status (what has happened) and instructions (what to do) – in consistent	Minor	4
11	Main Screen	“Your exec is available grids window” message requires user to know of remote home path and open it themselves	Minor	6
12	Services/Applications	After dragging the exec file the button changes – strange fonts and difficult to read	Minor	4,1
13	Main Screen	Replication of output in My Workspace area	Cosmetic	1,4
14	Submit source code Dialog	Submit source code dialog box remains open after successful compile	Cosmetic	1
15	Overall	Message “executing remote application” is confusing – thought we were executing my code	Cosmetic	4,2
16	Overall	Output reporting of compilation is wiped and replaced with output from execution – would prefer it to all remain	Cosmetic	4

Table 5.2 continued: Heuristic Evaluation Sheet 1(Usability expert)

5.1.4 Results from three domain Experts (Work-Domain Expert Analysis)

This evaluation was conducted by the author, the usability expert and the three work-domain expert. The usability expert took about 10 minutes to explain the usability basics [102] and ways to identify a usability problem to these users. One user was nominated to go through the task and perform the steps. In this case, it is interesting to note that the detailed task given in the other two evaluation processes was not given. This was because we wanted to see how a user interacts with the interface without any guided steps. All that was given to the user was a general overview of the particular task that was going to be performed by the interface. The nominated user tried to perform the task while the other users observed and made comments. When the user finished, the three users then went through the process all over again but now giving verbal comments of the usability problems. The usability expert then categorised these problems and this is reported in Table 5.3 in a heuristic evaluation manner.

Problem Id	Location	Description	Severity
1	submit source code dialog	"input file" expecting a browse button but asks for name - get some lengthy horrific file names unable to type them out	catastrophic
2	submit source code dialog	restricted to single source file and input file - would expect option to add multiples	catastrophic
3	Overall	no indication of what had happened and what to do next, no clear indication of sequence	catastrophic
4	Overall	no feedback on what is running	catastrophic
5	submit source code dialog	needing to know the name of the data file before compilation means you have to recompile every time you run it,	catastrophic
6	buttons at top	"submit job", "submit source code" etc difficult to read, especially "drag your input file" to here because of fonts	major
7	submit source code dialog	didn't know what an input file was- what was being referred to	major
8	login	login button didn't disable so didn't know if you have done anything, dialog box just sits there with no progress, no log out button, no indication that you are logged in	major
9	file browsers	way too small - cannot see the files you need - applies to all of them	major
10	services/applications	takes too much space	major
11	main screen	"submit source code" is not leaping out as first thing to do, think might have to select file in workspace first	major
12	data repository	output file goes to same folder as input data file but display does not refresh to show you this	major
13	overall	if it is a large output file would not want to have automatic transfer back to my machine - lots of users could stumble on this	major
14	submit source code dialog	"type in your source code" tempting to type in source code - labels are confusing - "select file"	minor
15	grid proxy unit	correct action is "create" but expect to see "login"	minor

Table 5.3: Heuristic Evaluation Sheet 2(Work domain expert)

16	main screen	resize handles but they don't allow resize	minor
17	main screen	styling of "submit job", "submit source code" etc means they don't look like buttons or at least active buttons	minor
18	submit source code dialog	expect it to close automatically after successful compile, not sure if "close" would cancel the job	minor
19	services/applications	unclear what the difference is between "application" and "exec", once it is compiled they are the same thing	minor
20	services/applications	strange text box that you can't do anything with	minor
21	main screen	data repository & my workspace are both on local machine - this is unclear	minor
22	overall	no feedback to say that the code is running - the button doesn't change and nothing in the output log	minor
23	my workspace	cannot open the output file from My workspace	minor
24	my workspace	message about the output file could be in the output log	minor
25	main screen	confused what is the difference between "submit code" and "submit job"	minor
26	services/applications	after you have run a exec the button doesn't change back to drag an exec file	minor
27	my workspace	the output file message in a box - no label, didn't know what it was	Cosmetic
28	overall	would make more sense as a one step process - give it all parameters and do it once	Cosmetic

Table 5.3 continued: Heuristic Evaluation Sheet 2(Work domain expert)

Assuming there were no time constraints, the usability problems identified by this particular evaluation would be used to redesign the interface because it captured the opinion of three domain expert users with a usability expert present and they were already briefed on usability principles so this could really go in for a heuristic evaluation which really needs at least three users.

5.1.5 Validity of the Results.

We believe that results obtained from this evaluation are a true representation of the evaluators' experiences whilst interacting with the built interface. As stated earlier a usability expert was present to conduct the evaluation and the proper guidelines and methods for this particular situation were adopted. We believe that given time a repeat evaluation with different set of users could give rise to more usability issues not

captured by the first evaluation. We also believe that the original users would have been more desirable for the evaluation so as to ascertain that what they required from the system was met.

In the absence of original users, domain experts and usability expert evaluation of the system has produced results that have demonstrated that our objectives have been met. We have learnt in the process of gathering these results that users requirements would be accurately captured if they were able to participate in the evaluation of the interface from the early design stage to the final product stage.

5.1.6 Usability problem of the interface

There was a comparison analysis from the three results gathered from each evaluation. The usability problems are summarised starting from the most critical to the least critical (see Table 5.4). As observed from the evaluation process, some of the problems are minor and will not require too much effort to be corrected. Others are major and would require some bit of coding to be corrected. Solution to some of the problems are difficult as the interface used some functions from the Cog toolkit which was not designed by the designer of the interface but was useful for the interface to show how an average user will have access to the underlying Grid network.

Comparison of the heuristic evaluation performed by the usability expert and the group based domain expert evaluation produced common problems identified by each evaluation. Some problems identified from the questionnaire also were common with those performed by the group based domain evaluation. The group based domain evaluation results in the most relevant here and is discussed in table 5.4.

ID	Usability problem	Action to be Taken
1	Dialog button for login to Grid confusing and not consistent with interface.	This interface is embedded from the cog toolkit but it was necessary to be used for testing purposes.
2	Submit source dialog to be closed after executable becomes available.	This can corrected easily.
3	Remapping the overall process so that the flow of action is more of wizard, which can be useful to novices.	There is no time to correct this but the interface was built to have each process independent of each other to make it more flexible.
4	Login button did not disappear after logon and no message to confirm user logged on.	The message to confirm user logon can be sorted but the button was embedded and could not control its closure from the Grid user interface.
5	'Type in File' in submit source dialog should have a browse button and will be good to be able to put in multiple input file.	This interface was meant to demonstrate the usefulness of an easy interface and so a single input file example is used for now.
6	The output file is not reflected in the data repository window.	This is a programming problem and can be sorted.

Table 5.4: Group based domain evaluation

5.1.7 Requirements specification for the Users

In chapter three, we identified the requirements of the interface being built and we have summarised the list of these requirements that have been met in Table 5.5.

5.2 Comparison of Grid User interface with other approaches.

We believe that we have designed an interface that aims to present to users an easier way to access the Grid for everyday use. The interface is simple and users can understand what it aims to achieve and have an idea of the task they can perform with it. Other approaches looked at, in the literature review have focused more on functionality rather than usability and we believe that we have been to prove that it is possible reduce the steps taken to grid-enable an application, present an interface close to a users working domain and grid-enable an application is an effective manner.

LIST OF REQUIREMENTS	COMMENTS
Users want to submit their existing code via interface without modification	We were able to achieve this with a sequential c++ code
The user wants to have their Job run successfully.	We were able to successfully run a source code but a compiled code did pose some problems.
The user wants to recover from errors easily.	The interface was designed in such a way as to avoid errors.
Users want icons that depict the actions to be performed.	We believe some icons were self explanatory but the evaluation process pointed out that this could be improved.
User wants useful feedback and Monitoring throughout interaction.	Through out the Job submission, user was able to view all actions taken place. Users advised it would be useful to have a log of this action for later references.
The interface will allow users perform drag and drop.	This was achieved with the demonstration of a source code job execution
User wants some Grid 'commands' represented with Buttons on the interface.	Not all commands were implemented but the authentication process and submit processes were encapsulated in buttons.

Table 5.5: Users requirements review on Evaluation

5.3 Conclusions

As discussed in the above sections, evaluation was conducted in three different ways. Each of the evaluation proved useful as it helped to identify the potential usability problems that can occur when the real users of the Grid interact with this interface. This evaluation also buttresses the point that usability experts and domain users (Users with good computer knowledge) can help determine usability problems in an interactive system.

The evaluation process hoped to confirm that the interface will present to the user an interface that has the following features:

- Ability to drag and drop.
- Effectively Grid-enable an application.
- Reduce steps needed to Grid-enable an application.
- Designed for both experience and novice users

We believe that this interface has met this aim even though from the evaluation conducted, a lot of usability problems were discovered. Given time, it would have been possible to redesign the interface based on the usability problems that are most critical from the previous evaluation and then conduct another usability testing with a different set of users to see whether the initial problems have been fixed and new ones discovered.

6 Conclusions and Further Work

6.1 Conclusions from experiences

This thesis has taken into account diverse technologies involved with building Grid applications. The most important aspect of this report relied on a lot of Grid standards, which is currently dynamic. Fortunately web services are standardized and as such were the starting point for this project. Further exploration come to a decision that a new way to Grid-enable an application had to be designed so as to satisfy the hypothesis of this work. The interface was designed and tested and the results analysed for further work. The GGF and the National e-Science centre [78] have continued to initiate conferences geared towards setting the right standards.

The literature review explored the Grid in general and investigated other technologies that existed before the Grid. The Grid middleware was looked at as that is the framework to which most programmers use in programming the Grid. Web service technology that preceded the Grid technology was thoroughly investigated. Grid projects were thoroughly reviewed and analysed. This review helped to categorise the approaches in building this projects under two broad headings.

This project is involved in a technology that is rapidly evolving and as such a lot of complexities were involved in deciding the right step to take to tackle it. The two main approaches to Grid-enable an application were explored. The application-based approach was chosen. The different ways to implement this approach was also explored and the application code approach was chosen. Specific users were interviewed to see how they currently use the Grid and a new way was proposed for using the Grid interface tool to make it easier and efficient to Grid-enable an application.

In the design of the Grid user interface, we explored the various issues that involved most especially, library issues. This proved to be an important aspect of the Grid-enabling process. We were able to develop a library verification process that would help for a successful Grid-enabling process and the limitations that were presented by the existing system design.

6.2 Further work

This project was embarked upon, with the hypothesis that we could reduce the steps taken to Grid-enable an application and do it more efficiently. In gathering the user's requirements and establishing user tasks and use cases, it became evident that doing this will involve further investigation on the understanding of how libraries are handled. A review of previous Grid projects gave an insight to why these projects are implemented without much care to how the target users currently work. This is due to the difficulty in handling libraries that are needed by most applications for successful execution. This project has been able to show this problem exists and more efforts need to be made on the best way to tackle it.

The start of the project was influenced with a need to handle scientific users who would normally have complicated codes to deal with. The testing bed would have been the white rose grid as it is one of the Grid centres where users would normally request to use Grid resources. The White Rose Grid (WRG) has in its underlying Grid environment different machines and different platforms and diverse users with different programming needs. Since this interface is being intended to be deployed on the WRG, it would have been ideal to design it to cater for most of the services that this Grid centre offers.

First and foremost it was not possible to develop and test the interface with the White Rose Grid for safety and policy issues. To simulate a storage resource broker, it was decided that a user's drive at the remote end be mapped on the client end and this can be viewed in the Grid interface. This was not possible on the WRG for security reasons. Hence we moved to the virtual machine option that allowed us to simulate the white rose Grid to some extent. With this option, we could explore and try different methods and techniques to enable us to achieve our aim without fear of crashing the network. Secondly, only one machine was used in this test bed as opposed to several. The Grid interface built considered a sequential C++ program to demonstrate its usefulness but it can be extended to cope with other programming languages. The interface would then need to give the users options to see the compilers available at the remote Grid network and choose the compilers they require for their code. The library verification process proposed can also be extended to

provide a periodic way of comparing remote file library files with library files on the client machine.

One of the ways for dealing with the library issues was to copy the libraries to the user remote directory. This option given time could eventually clog up the remote machine and it may become difficult to manage different libraries for different users for different codes. It would be useful to really concentrate on making sure that the library verification technique is improved so as to avoid having duplicate libraries at the remote end.

The endianness representation of computers still remains a pressing issue and it would be very good to have a general method or algorithm of dealing with endianness of the sending and receiving computer. This particular problem does not fall within the scope of this project but it would be very useful if further research in computer architectures could provide a standard endianness and then we don't have to worry about how data would be represented on computers whenever there is a transfer.

The Evaluation process was very revealing and useful. Domain experts and usability experts were used and a lot of usability problems were discovered with a severity of minor to catastrophic. Even though there has not been enough time to redesign the interface for another evaluation, we are confident that this project will give useful information and guidelines to how Grid applications are built with an emphasis on the target users of the interface. This project has also investigated the issues involved in the usability of a Grid-enabled environment. The Grid user interface that has been built in the project can serve as framework for an intended Grid application developer.

6.2.1 The Grid User Interface

The Grid user interface that was designed was semi-functional aimed at proving our hypothesis. But further work would concentrate on the functionality as well as the usability of the interface. The four windows designed so far have limited functionality due to the time constraint but we summarise below what each windows can present to the users in future:

The My Workspace Window

This window at the moment presents to the user the output of his result. The data store in the window can be further designed to allow the user drag input and output data.

The Services/Application Window

This window allows a user drag in their recently compiled source code or transferred compiled code ready for execution. Right now users are only able to run their own code at a time. Further work will involve the window being able to dynamically display a node for each Grid services added to it and able to display more than one Grid service at a time. It will also present to the users the wsdl of each service when users right click the service button so that users can use it properly and allow users to run a service and their own compiled codes at the same time.

The Available Grid Window

This window is currently presented to the user as a mapped network directory that allows a user compiled code to be displayed. Future work will have Grids displayed in boxes with the services that each Grid provides in each box. This will be automatically refreshed like every 1 hour so that users are always up to date.

The Data Repository Window

This provides all the storage nodes that are available to the Grid but this has used the normal file system format due to constraints. The best way to present this store will be to use a storage resource broker (SRB) that can present to the user a global file system directory in a logically distributed manner and allow new data to be automatically displayed. Presently, Grids are being translated into clouds but even this comes with its own set of problems.

Appendices

Appendix A: DAME: A scientific developer experience of building a Grid environment.

Mark Jessop who was part of the DAME project has composed this report. It is used in the report to give an idea of how a scientific developer might build a Grid environment today.

The DAME project was undertaken as an e-Science Grid pilot project under the UK's Engineering and Physical Science Research Council (EPSRC) funding, set up to demonstrate the benefits of Grid computing in engineering applications. DAME was a £3 million, 45-person-year project, which completed during 2005. It had three commercial collaborators: Rolls-Royce Group plc., who provide the engine data, information about the problem domain and the definition of the virtual organisation, Data Systems and Solutions LLC., who supply the data management frame work and Cybula Ltd. who are commercialising the underlying AURA (Advanced Uncertain Reasoning Architecture) search technology on behalf of the University of York; along with four academic partners, The University of York; The University of Leeds, The University of Sheffield and The University of Oxford

The DAME project investigated the problem of building a Grid based diagnosis and prognosis system for aero-engines {jet engines}. The focus of the project was the analysis of vibration data collected from aero-engines in order to undertake diagnostic and prognostic processes. Such engines will typically produce around 1GB of vibration and performance data per flight. Therefore an aircraft carrying four engines will generate 4GB of data per flight. At the fleet level, terabytes of data are produced on a daily basis.

Through the use of an on-wing diagnostic capability, an abnormality could be detected in an aircraft's engine data. Part of the diagnostic process required the searching for similar behaviour to the abnormality in engine data from other aircraft.

DAME aimed to build an end to end workbench environment for; data collection and management, data processing, engineering tools, result correlation, and knowledge capture, linking these elements together and extracting the generic properties for use in other problem domains. This was successfully achieved as the DAME model has been used in both a commercial applications and further research projects.

The scenario developed was that data would be downloaded from an aircraft when it landed. The data would then be processed by systems similar to the on-wing diagnostics. It was envisaged that the on-wing systems would be hard to update, so more intensive and up to date processes would be available on the ground. Once the data had been processed, the results would be compared with the on-wing systems to generate a fault report. This would define whether or not an engine required intervention by an engineer. In the case where no abnormalities were found, the normal turn around procedures for an aircraft was followed.

When abnormalities were detected, the DAME system attempted to identify the probable causes and solutions by searching the fleet archive of engine data for similar abnormalities. The results of this search were used to generate a list of similar events and therefore the probable cause and a diagnostic process for the engine maintenance team to follow, in order to confirm the diagnosis.

The system allowed engine events to be escalated from maintenance engineers through various levels of expertise to aid in the diagnosis.

It was intended that by identifying failure signatures, it will be possible to spot the beginnings of failure modes before they became critical. This would allow the airlines to more accurately schedule maintenance and avoid the situation where aircraft have to be unexpectedly removed from service, and therefore avoiding the huge costs and delays that this involves.

Challenges

A number of key challenges were identified:

- Understanding Grid Technology
- Data Management

- Data Search

Grid Technology

At the beginning of DAME, Grid was a new and fast emerging technology that was picking up a considerable pace. The Global Grid forum had only recently formed and the UK was just starting to push into the world of Grid, hence the pilot projects. The entire DAME team was new to Grid and the White Rose Grid was still in the early planning stages.

Data Management

The large volumes of engine data to be collected presented a particular challenge since it would be downloaded at airports all over the world. This data had to be entered into the DAME system and processed in a timely fashion to allow aircraft to be turned around within schedules. Although the DAME system was not to be a safety critical system, it was still envisaged that it would affect whether or not an aircraft was allowed to fly.

The challenge was to build a big enough data repository that could accept data at the rate at which it arrived and present this data for processing when required and be able to provide results in a timely fashion. One of the key questions to be answered was where was the engine data would be stored, either where it was collected or brought together in regional or central data repositories. Both possibilities presented their own problems such as available network bandwidth, processing capability and redundancy, all with relation to cost.

Data Search

The search process required that a vibration signature be compared against the fleet archive of vibration data. In order to achieve the aircraft turn around times this search needed to be extremely quick. The results of the search process were in terms of vibration patterns, these had to be turned into aircraft event centric results. A case based reasoning system was needed to perform this mapping.

Grid Implementation

The basic requirement for the DAME system was that large quantities of data would be arriving quickly at very widely dispersed locations. This data had to be archived and made available for processing. There were two sides to the nature of the processing. Firstly a batch process and set of services would be run over the data to generate a basic diagnosis. This would be the same for all new data items. Secondly, users or domain experts may request ad-hoc processes/services to manipulate the data when investigating a newly observed problem.

It became apparent that it wasn't going to be feasible to collect all the data in a central processing location, the network requirements alone would be prohibitively expensive, and this wouldn't necessarily utilise any Grid elements.

The simple solution was to leave the data where it was collected, and have a smaller/cheaper set of resources at each location to perform data storage and processing. This would allow the sites and users to be connected via a lower capability and cheaper network. This is also had the advantage of supporting the Grid paradigm; a dispersed set of resources with no central control, bought together through common and open protocols to provide services.

The implementation needed several Grid tools to enable the system to be built; a data storage system that could be managed in a distributed fashion, some way of invoking services and processes, some knowledge capture system and a user interface.

Grid Technology

At the time that DAME was getting going, the Globus Toolkit was tool favoured by the UK Grid community, so this was chosen for implementing the processing Grid infrastructure. The toolkit provided a mechanism for linking together compute resources and allowing processing jobs to be submitted for execution. Around this time, a service orientated architecture (SOA) for grid based on web services was being developed by the Globus team, and this formed the basis for the latest ,at the time, version of the toolkit, GT3. This was the first really useable version of the

toolkit and encapsulated the open grid services architecture (OGSA) that formed the basis of future Globus work.

Globus was installed across the White Rose Grid to allow grid services to be accessed from all partner sites. Batch jobs were also submitted via Globus, but discussion of this is outside the scope of this document.

Data Storage

The data storage element was provided by the SDSC (San Diego Supercomputer Centre) Storage Request Broker (SRB). This is a client server system for joining together distributed storage resources into a virtualised storage system. In effect each resource provides a physical location where data can be stored. The SRB system joins these together and provides a virtualised view of these resources such that virtual directories can contain files that whilst appearing to be in one location, are in reality located on different physical locations. Access to data is through logical path names, without any need to know the actual location of the data.

SRB was installed across the White Rose Grid, with each site (York, Sheffield and Leeds) representing a separate data centre (airport).

Data Search

One of the research issues for York was the searching of jet engine data using the AURA technology. An output of this work was the Signal Data Explorer (SDE) tool for exploring and searching time series data. However, the SDE was a desktop application and unsuitable for working with fleet archive volumes of data. A distributed search process was built to manage the large volumes of data, with SDE as the front end.

This work generated a system called the Pattern Match Controller (PMC). This deployed a grid service at each data site coupled to a local search engine. The search engine only worked on data stored at the local data repository. A client application, e.g. SDE, connected to a single PMC grid service and requested that the distributed dataset be searched. This PMC communicated with the other sites to search all the data. At no point did any data leave the local site. All data access was through SRB.

The results consisted of a set of ranked pointers to files stored in the SRB system. The user could view each result in turn, which caused the matching data to be retrieved from SRB. In reality, although the search engines returned many results, the user only ever the top few, meaning that very little data actually had to be transmitted.

This demonstrated the power of the SRB system; data could be locally addressed without needing to know about any other sites. However, at the user level all data was simply seen as being part of a single repository and could be accessed without needing to know where it was located.

Experiences

As stated above, initially Grid technology was relatively new and moving at a fast pace. At the start of DAME Globus Toolkit 2 was the state of the art, with the previous versions being esoteric and widely accepted as near impossible to work with. GT3 followed fast on the heels of GT2 was the first version that did what it was supposed to. However, the documentation was almost non-existent resulting in very slow progress.

The OGSA architecture was very heavily layered and particularly difficult to work with. Development of basic services took several months to complete, and a full understanding took longer. Developers found that the layers tended to obscure the nature of failures, making the debug process very difficult. More often than not, it was actually the process of deploying and securing services that caused problems, rather than implementation errors in the services themselves. It was found that the configurations had to be “just” right before any success could be achieved. Replicating this and providing a coherent environment for services across the White Rose Grid presented a particular challenge.

The fast pace of toolkit development kept up throughout DAME, within the first two years, three separate versions; GT2, GT3 and GT4 appeared, all of which were different. Although GT4, which is still the current version, was still SOA based, it

changed the underlying service model and architecture, resulting in services developed for GT3 being incompatible.

In the early days of DAME, SRB was in a similar state and was discounted as a solution to the data management problem. It wasn't until the end of year two, that it became a useable technology and DAME decided to adopt it. SRB suffered from similar problem as Globus, mainly the lack of coherent documentation making progress slow and hard work. However it has been proven to be a relatively simple technology and performing as expected.

From an application developer's perspective, Grid is not easy and this hasn't changed in the last five years. Building a grid based SOA requires a good grasp of the Globus Toolkit and it's concepts. It is in the debugging and deployment of service that requires the greatest level of understanding, and this is not suitable to those that want to quickly put together a service.

Simple batch processes are a considerably easier to build and deploy, although developers need to think about which libraries and technologies are available on the target end points.

SRB was a considerably easier tool to work with than Globus, partly due to it remaining stable through out the project. In effect the SRB team wrote the documentation and then set about implementing it. That meant that if a feature in the current version did not exist, it might well do in the next one. The documentation was still weak, but once the initial learning curve had been conquered either using or building applications against SRB was quite easy.

Conclusion

DAME built a considerable Grid application consisting of many tools, services and batch processes, some of which have been described here. The development process was difficult and time consuming, mainly due to the lack of documentation and the pace of change associated with the Globus Toolkit.

It is definitely the case that building Grid application is not trivial or for the faint hearted. It would not be unreasonable to expect potential Grid application developers to expend considerable efforts before being able to deploy their own applications

Appendix B: Conferences and Seminar

There were a lot of conferences and Seminars organised by various Grid communities and this has proved very useful, as other researchers doing similar topics have shown progress on their work. This proved very useful for this research topic and I have been fortunate to give a presentation at the University of Durham. I was able to present a poster at the All hands meeting 2008 and the abstract of my PHD received positive reviews.

E-science Workflow Services

Date: 3 Dec 12:00pm – 5 Dec 3.30pm

Venue: e-Science institute
15 south college street
Edinburgh

Organisers: Dave Berry (National e-science centre), Savas Parastatidis

The conference brought together international researchers and locals all working on workflow products. This conference geared me towards the right track in presenting my departmental seminar that was held on the 17th-Dec-2003. All the workflow projects used in this report were all presented here and it was useful to meet the main individuals involved.

The 3rd North East Regional All hands e-Science meeting

Date: 10:00 – 16:30 Friday, 9th January 2004

Venue: Lindisfarne Conference Centre
St Aidans College
University of Durham

Organisers: Jie Xu and Rob Smith

This was a meeting that involved different topics like Systems dependability, Security, Developments and Applications. Of most interest to me was the Service level agreement in a service-oriented architecture. This is directly related to my project area as it entails the tools needed when a client needs to negotiate the use of a

resource. The main achievement was the fact that I was able to give a talk on the departmental presentation that I prepared in December.

10th Global Grid Forum

Date: 10:00 9-Mar – 17:00 14-Mar 2004

Venue: Humboldt University
Unter den Linden
Berlin, 10099, Germany

Organisers: Alexander Reinefeld

This seminar is organised by the main organisation driving the standards for the Grid. The seminar was a very beneficial to me because a work group was formed to look at designing API for the Grid. Simple API for Grid Applications Working Group (SAGA-WG) (<http://forge.gridforum.org/projects/gapi-wg>) will seek to hide details of service infrastructure that may exist to implement the functionality an application developer needs to build a grid enabled environment that is easy to use. The one but next one GGF12 would be giving progress on how far they have gone on this. It was a good opportunity to meet face to face with Steve Tucker a director of the GGF. I look forward to more close contact with him as time goes on.

North Eastern Regional e-Science Centre Summer School

Date: 10:00 – 17:00 26 July 2007

Venue: University of Newcastle
Newcastle
United Kingdom

Organisers: Hugo Hiden

Mode of Attendance: Presenter (Talk)

This seminar was organised as a summer school for PhD students as an opportunity to present work and meet and interact with other local PhD to discuss the our research. I gave a presentation of my PhD work so far. My presentation title was Improving Usability in a Grid-Enabled Environment

All hands meeting 2008

Date: 8 – 11 Sept 2008

Venue: University of Edinburgh
Edinburgh
United Kingdom

Organisers: National eScience Centre

Mode of Attendance: Presenter (Poster)

This conference is organised by the national eScience centre which aims to bring researchers that work in the area of Grid computing. Recently international researchers have also presented papers and talks. I presented a poster of the overall summary of my PhD.

Appendix C: Setting up the Grid User Interface.

Initial Setup

The initial setup involved my system pc100 acting as a windows client and ua026 acting as a Linux server with Globus installed. We had firewall issues on the network and as such we could not log on to the ua026 easily as we thought. We installed a virtual machine(VM) on ua026 with the same installation as the physical one. We then installed an interface on pc100 that allowed us to access the VM on ua026 and then we were able to submit jobs for testing whilst developing the Interface on pc100. Developing from a windows operating system was very simple and straightforward but as we got deep into the project we were unable to properly implement some aspects of the interface on windows so development moved to a Linux machine. The main aspects of the interface but influenced this decision was the library verification process. It was necessary to get the libraries that a user application/code depended on to run and this was not easily retrievable in windows. Linux gcc tool comes with the ldd tool that captures the libraries. We then manipulate the libraries for verification, updating, copying and caching. Deploying the already written code in windows to linux was straightforward as Qt is a portable platform independent tool.

Final Setup.

The initial setup described above all consisted of physical machines and that meant that it was not possible to development outside my work area. We then decided to explore the virtual machine option. We created three virtual machines: One for the client windows interface, the second for the client linux interface and the third for the linux server interface(with Globus installed). With this arrangement, development took place anywhere. This option also made us have a firm understanding using virtual machines as alternatives for difficult situations. This setup though with the use of a virtual machine is the same to setup a new user for the white Rose Grid. The figure below shows the final setup

Setup of Remote Virtual Machine

Installing and setup of the remote linux virtual machine (*agah-desktop*) was very challenging and cumbersome. The virtual machine was created with the necessary memory for the harddisk, RAM etc. Then we installed Globus on the machine. The Globus installation is credited to Aaron Turner of the ACAG group. Globus

installation was very time consuming and challenging and the most important aspects itemised below.

- Install Globus was installed in the /usr/local/globus-4.0.4
 1. While logged on as a super user, unpack the compressed **tar file: tar -jxf gt4.0.4-all-source-installer.tar.bz2**
 2. Change directory to the distribution cd gt4.0.4-all-source-installer
 3. Define GLOBUS_LOCATION to point to the directory where toolkit will be installed: **export GLOBUS_LOCATION=/usr/local/ globus-4.0.4**
 4. Configure the distribution
./configure --prefix=\$GLOBUS_LOCATION --disable-rls
 5. Build the toolkit by running **make** then next run **make install** to complete the installation -
- Setup the Certificate authority. This allows creation of users that can later login remotely to submit jobs
 1. Setup the environment for the globus user
export GLOBUS_LOCATION=/usr/local/globus-4.0.4
source \$GLOBUS_LOCATION/etc/globus-user-env.sh
 2. Now run the 'simple-simple-ca' command to begin the setup process.
\$GLOBUS_LOCATION/setup/globus/setup-simple-ca
 3. The script informs the user that the CA information about the certificate will be kept in **/home/globus/simpleCA**
 4. You are then prompted to put in the unique subject name. The following provided: **cn=agah, ou=simpleCA-agah-desktop, ou=GlobusTest, o=Grid**
 5. Enter an expiry date in this case 1825days
 6. Enter an email address where certificate requests will be sent
 7. Enter the PEM passphrase and this creates the CA and tells you that a certificate authority with the subject **/O=Grid/OU=GlobusTest/OU=simplCA-agah-desktop** has been generated.

8. Creates the package and tells you where the private key is located and the public CA is located. You will need this information to setup the client end.

PrivateKey=/home/agah/.globus/simpleCA/private/cakey.pem

PublicCA=/home/agah/.globus/simpleCA/cacert.pem

DistributionPackage=/home/agah/.globus/simpleCA/globus-simple-ca-8b7155a1-setup-0.19.tar.gz. You will notice that a unique hash number for the CA has been created.

9. To complete the setup of the GSI software, run a script in directory **/usr/local/globus-4.0.4/setup/globus_simple_ca_8b7155a1_setup.**

10. Run the **setup-gsi** script in the above directory with the **-default** flag so that the CA we just created becomes the default certificate authority for certificates created on this node.

11. Once the setup-gsi script is successfully completed, it means the CA just created is installed and is the default for requesting certificates on nodeB

12. Run **grid-cert-request -force** and that created.

/home/agah/.globus/userceer_request.pem

/home/agah/.globus/userceer_request.pem

/home/agah/.globus/userceer_request.pem

- Request a host certificate on node agah-desktop. After making the request we will sign the certificate using the CA on the node. After the request is signed we install the certificate in the proper place on the machine

1. Setup up the environment like in the previous step

2. Run **grid-cert-request** using the **-host** flag to indicate the fully qualified name of the **node grid-cert-request -host agah-desktop**

3. This will create the following files

/home/agah/.globus/usercert_request.pem

/home/agah/.globus/userceer.pem

/home/agah/.globus/userkey.pem

4. Now that the certificate has been requested, the request must be signed by the CA. Setup the environment and use the command **grid-ca-sign** and when prompted enter the password for the CA

```
grid-ca-sign -in $GLOBUS_LOCATION/etc/usercert_request.pem -  
out $GLOBUS_LOCATION/etc/usercert.pem
```

5. IT then says the new signed certificate is at
/home/agah/.globus/usercert.pem

- Setup Remote Drive. For the test bed demonstration we used the smbclient service to access the remote drive of the Grid network. Smbserver service was installed on the Server end so that users are able to mount a remote drive from their client machines. We were also able to map the same remote directory on the windows client.

Setup of Client machine

As explained there were two client machines during the programming of the interface: The windows and linux virtual machines were installed like the one above. The following steps for a user to successfully logon to the Grid and submit job to the Grid is the same for both operating systems.

- Installing QT/C++ Compiler
 1. Install Visual studio in windows/Linux already comes preinstalled with the gcc tools
 2. Install QT in both windows/linux virtual machine and set the environment variable in windows to PATH=C:\Path to Qt\bin in linux to export QTlocation = \Path to QT\bin
 3. To start development in linux to to a shell prompt and create a project and then issue the following commands.
 - **qmake -project**
 - **qmake project.pro**
 - **make project.**
 4. To start development in windows, create a windows project folder eg project. While in command line,
 - **qmake -project**
 - **qmake project.pro**
 - **qmake -tp vc** to create visual studio project.
- Setup Globus
 1. Change directory to /etc/grid-security in remote file and zip all the files. Tar cvf /tmp/grid-security.tar grid-security/

2. Copy the files to the client side
3. Install the cog kit and set up the environment properly
4. From the command line run cog-setup and follow all the simple instructions until it asks for the key and put the CA hash=dc9d319e.0
5. A particular file (cog.properties) is created when cog-setup completes. The file is in the windows virtual machine directory C:\Documents and Settings\User\.globus and linux virtual machine directory \home\agah\.globus. The windows virtual machine content is in listing C1.
6. Test that we are able to mount the remote drive of the Grid machine agah-desktop from both the windows and the linux client machines.

```
#Java CoG Kit Configuration File #Tue May 20 13:24:11 BST 2008
usercert=C:\\Documents and Settings\\agah\\Desktop\\.globus\\usercert.pem
userkey=C:\\Documents and Settings\\agah\\Desktop\\.globus\\userkey.pem
proxy=C:\\DOCUME~1\\root\\LOCALS~1\\Temp\\x509up_u_root
cacert=C:\\Documents and Settings\\agah\\Desktop\\grid-
security\\certificates\\d69d319e.0
ip=192.168.248.129
```

Listing C1

The white Rose Grid (WRG) setup

As mentioned earlier, the programming development took place using the virtual machine setup. When the interface was developed to some extent, it was necessary to explore how users setup accounts on the WRG and how the submit jobs and the remote directories that they have access to. The following steps explains the process of setting up a user *agah* on the WRG

1. Apply via email to the National Grid Service for an science certificate
2. After 1-2 days you get a reply with the username and told to go to a link and use browser to import certificate. Firefox was used to import certificate
3. Aaron the WRG support engineer created a user account with username aeo101 and password –MOBpIcJP and told that my home folder is /wrg/home/csci/aeo101.
4. Installed cog on my computer and set the environment variable on the control panel.

5. From Firefox, go to the menu Tools->Options->Advanced->Certificate->Your Certificate and enter. Click on the eScience CA -> Backup and save the file as *agahnewcert.p12*.
6. You then copy this file to the WRG. Issue the command *pscp agahnewcert12 aeo101@pascali.wrg.york.ac.uk*, put in password and file copied
7. A Script(*pkcs2globus*) in remote end in */usr/local/bin* is run that converts the *pkcs* certificate to a *globus* certificate.
8. At remote end create a folder *.globus* in */wrg/home/csci/aeo101*, change directory to the folder and *chmod 700* and then issue command *pkcs2globus agahnewcert.p12* and this creates *usercert.pem* and *userkey.pem*. Then *chmod 644 usercert.pem*, *chmod 600 userkey.pem*
9. Copy from */wrg/home/csci/aeo101/.globus* to my computer on *D:\.globus*. While in windows, issue command *pscp aeo101@pascali.wrg.york.ac.uk:.globus/* . .*
10. Setup cog, put in registration details and when cog looks for the *usercert.pem* and *userkey.pem*, point to the appropriate folder *D:\.globus*. Signing policy is *367b75c3.0*.
11. WRG support engineer made sure the *globus* services are all working: Check the host certificate is valid and the gatekeeper is running. This took a whole day to sort out and the job submitted at remote end ran successfully.
12. Now run a job from the windows end to test. Ran *globusrun -o -r troilus.wrg.york.ac.uk -f troilus.rsl* and it was successful. *troilus.rsl*:

```

& (executable=/bin/date)
(directory=/home/csci/aeo101)
(count=1)

```

Listing C2

Appendix D: Screen shots of Grid User Interface

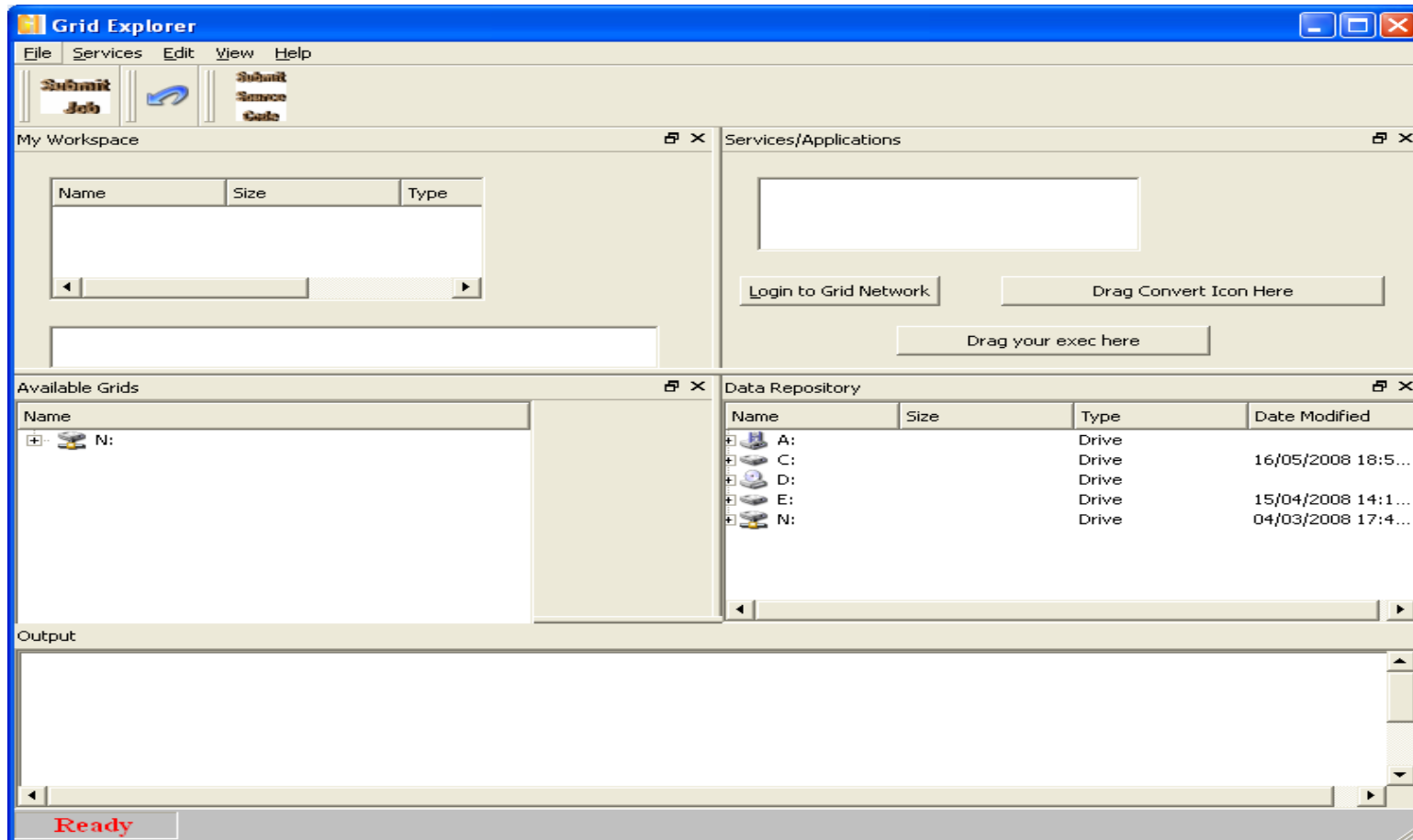


Figure 4.8.1: The Grid User Interface launched

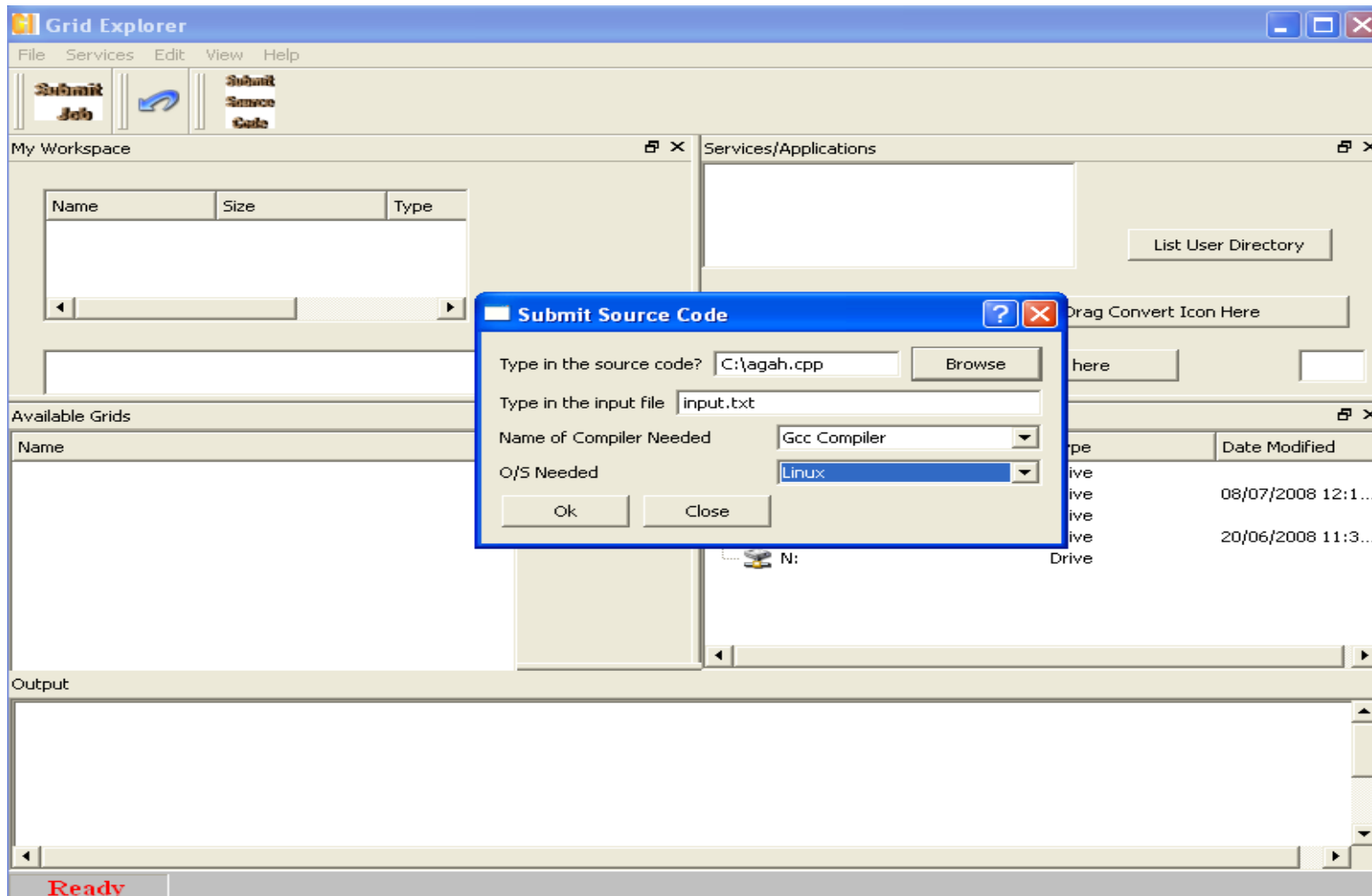


Figure 4.8.2: User tries to submit job without logging on

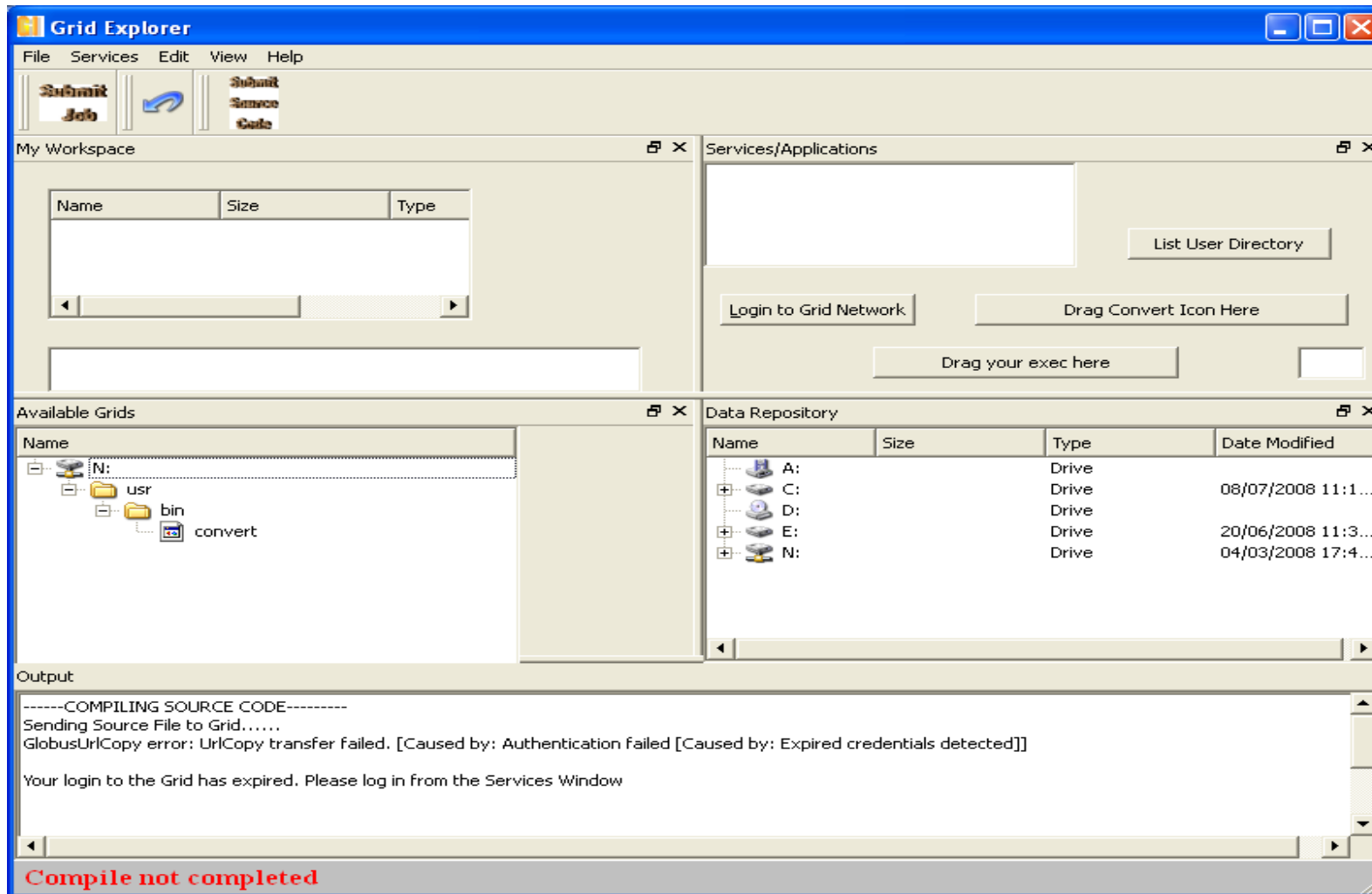


Fig 4.8.3: Users get error and is instructed to log on before continuing.

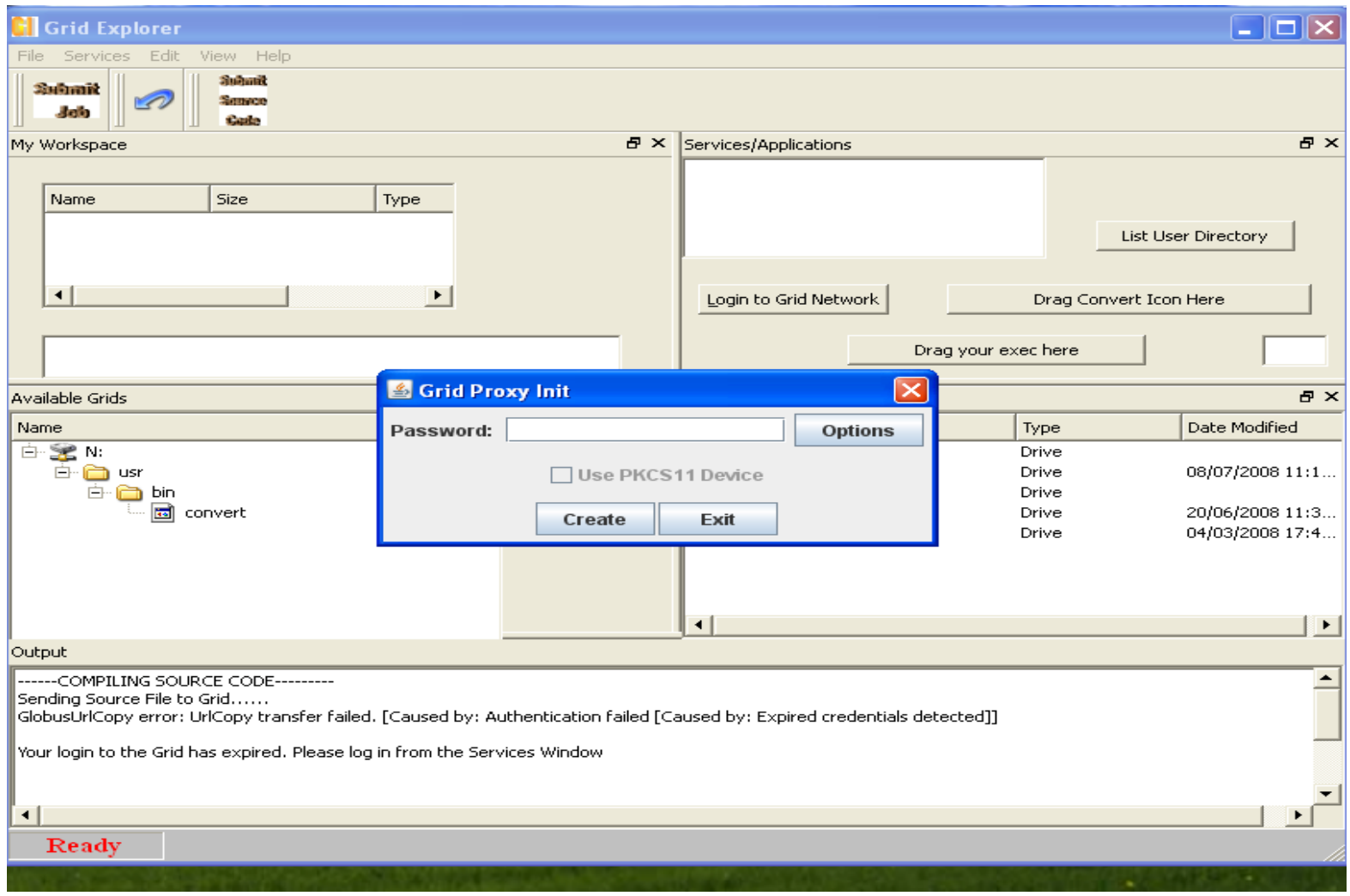


Figure 4.8.4: User clicks the logon button to log to the Grid

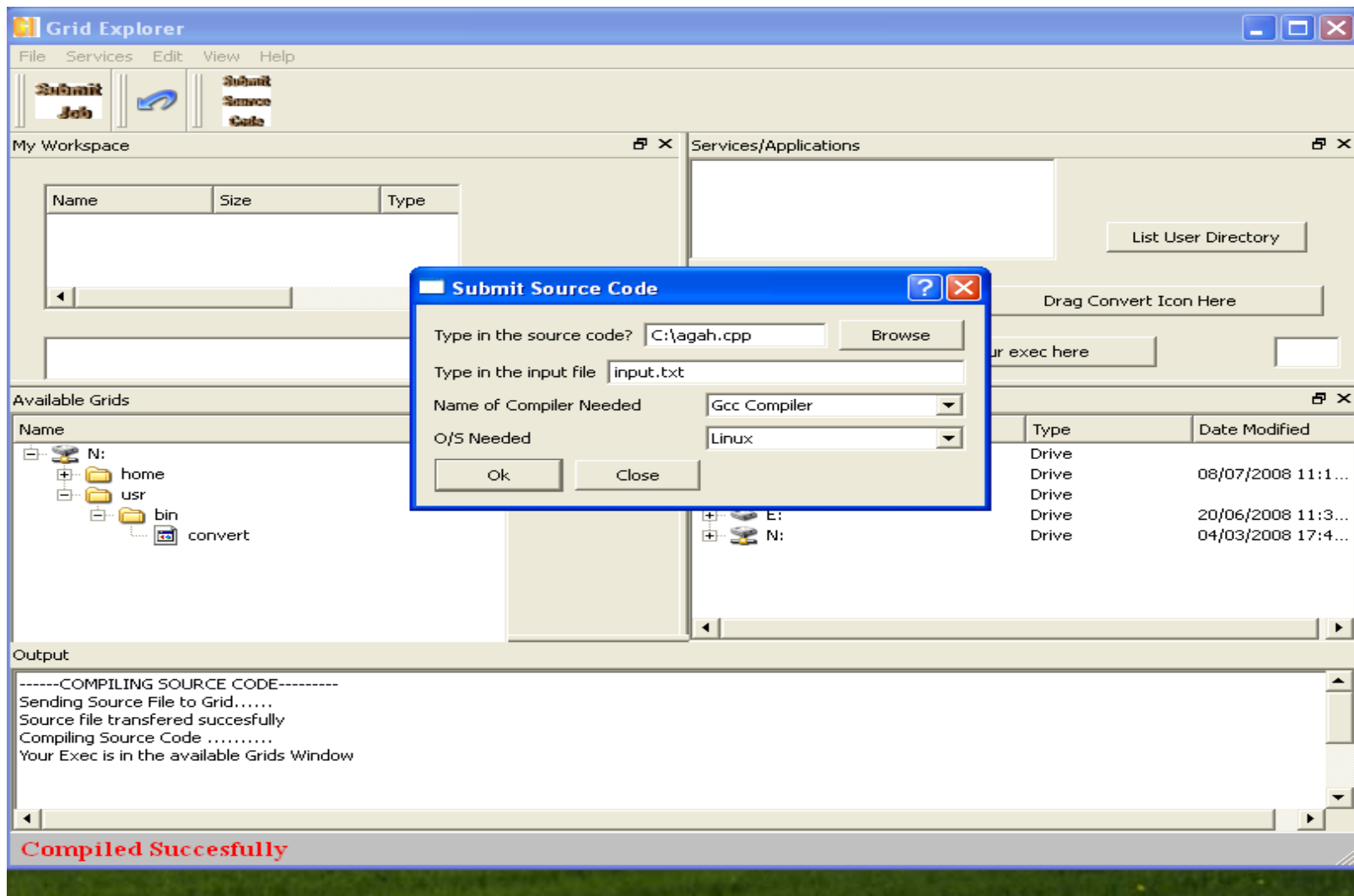


Fig 4.8.5: User submits a source code for compilation

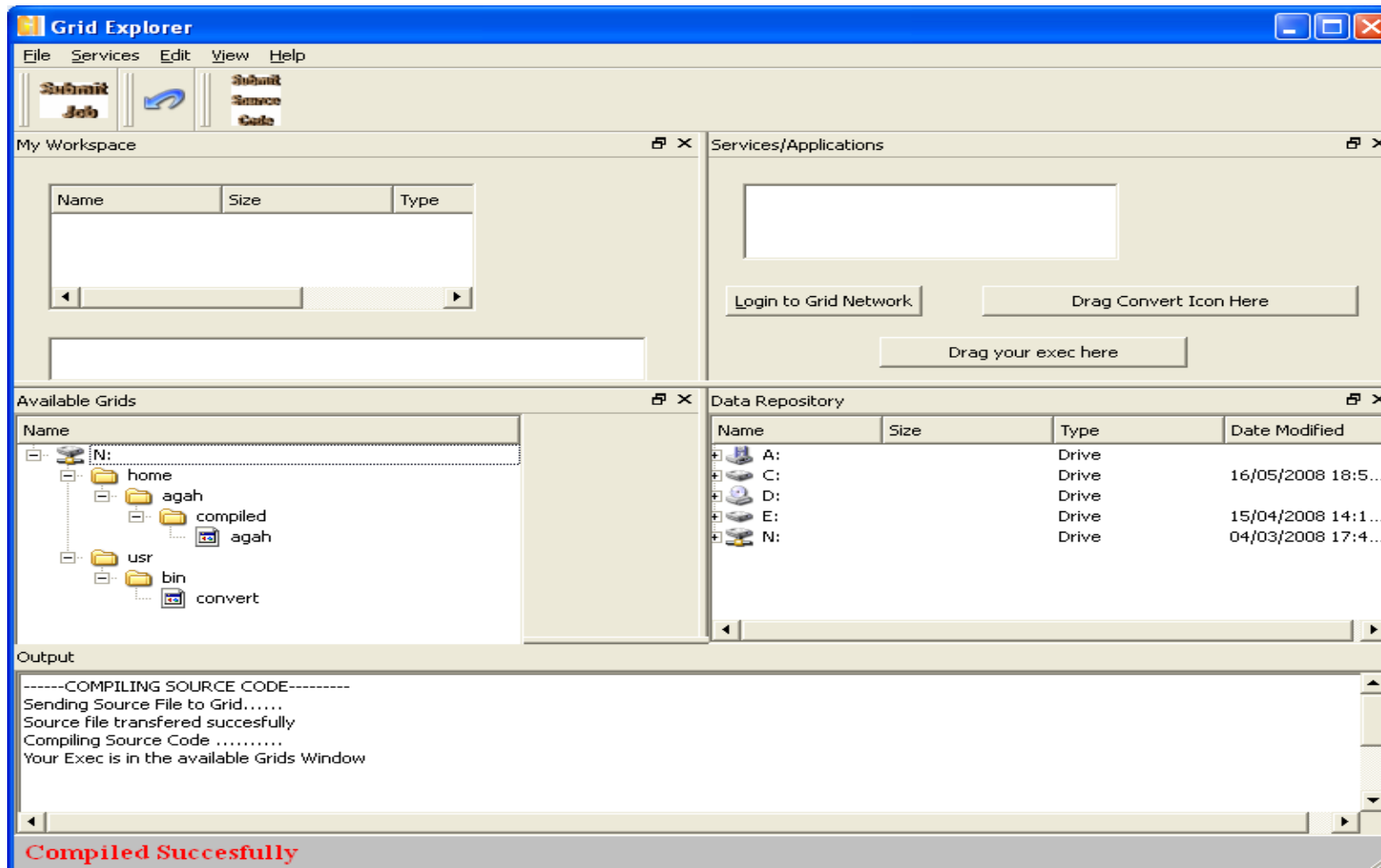


Figure 4.8.6: User closes submit source code button

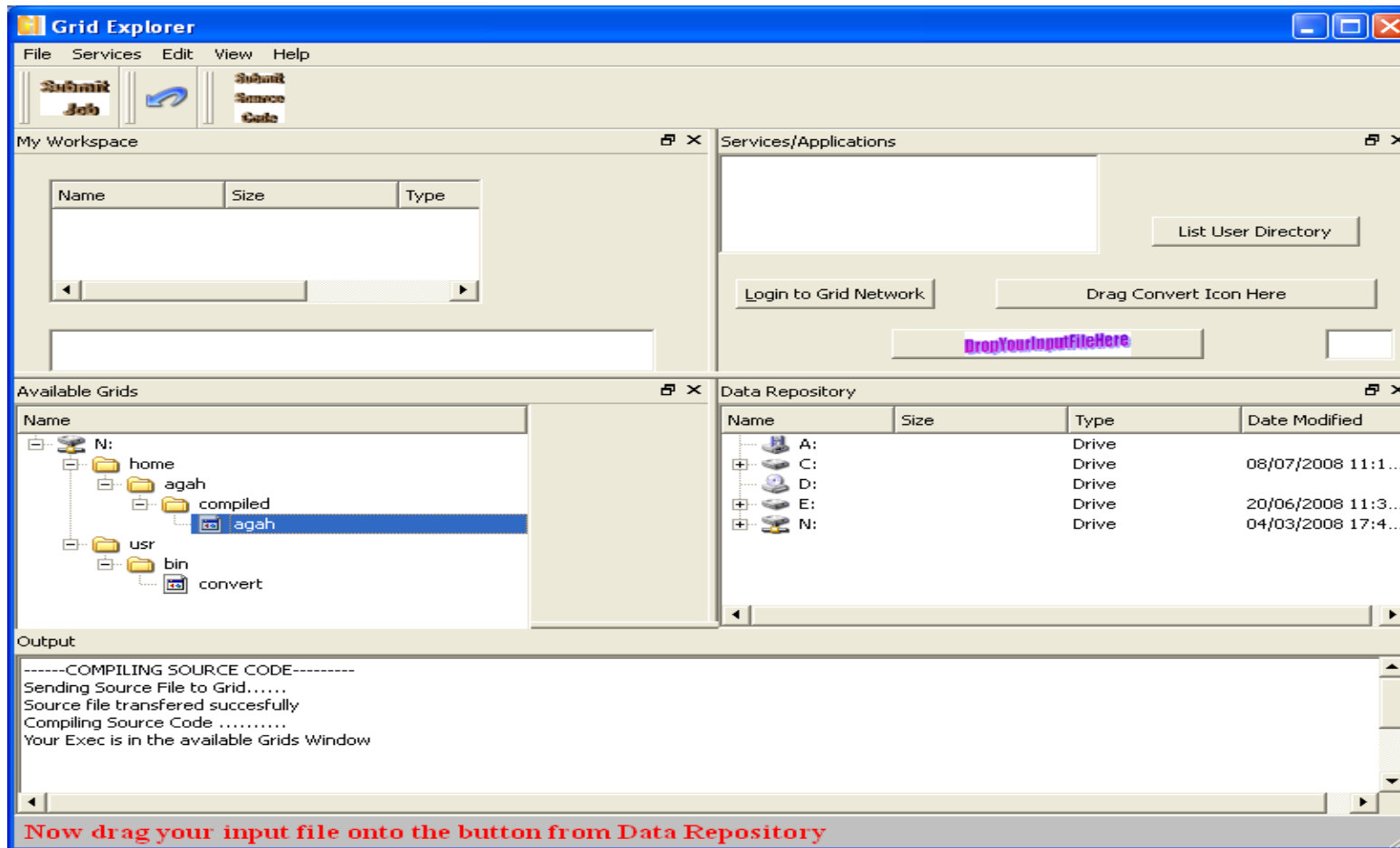


Figure 4.8.7: User drags the compiled code to the Services/Application window for execution

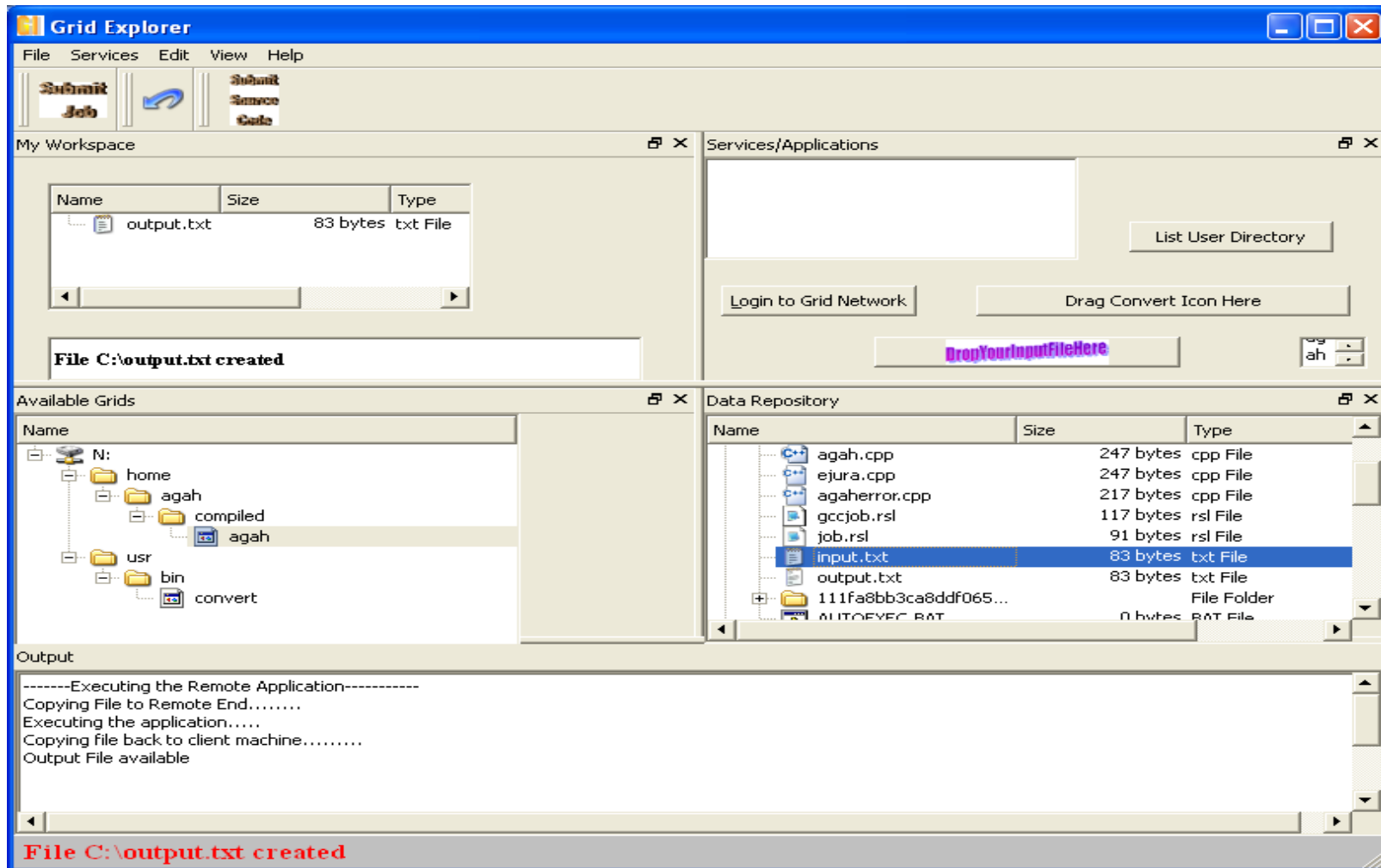


Figure 4.8.8: Users code is run successfully and the output displayed

Appendix E: User Interface Evaluation sheet.

Grid Explorer user study and expert feedback Directions sheet	
--	--

Introduction

Thank you for participating in this user study of Grid Explorer. Your expert feedback is much appreciated. We have tried to make this study as short and useful as possible. The purpose of this study is to provide feedback in the following three areas:

1. **Application:** how useful you think Grid Explorer is (to both yourself and an average Grid User).
2. **Ease of use:** how easy is it to use the Grid Explorer
3. **Adjustment:** your expert advice for modification / extension etc.

Privacy policy and your identity

At the start of this study you have the opportunity to provide your name. This information is not mandatory. If you do not provide these details your feedback will still be of great worth.

Grid Explorer: an overview

Grid Explorer is a User Graphical Application providing a front end to an underlying Grid network. The primary purpose of the Grid Explorer is to allow users to submit a source code to Grid network prior to execution and to transfer an already compiled code to the network for execution. In this particular evaluation, we focus on the steps taken to submit a source code to the underlying Grid network.

Items required

To complete this study you will be provided with the following three items (additional to these instructions):

1. The Use case Grid Explorer

2. The **source code, input file, O/S and Compiler** Information needed for the user study (/home/agah/agah.cpp, input.txt, Gcc Compiler and Linux respectively).
3. A Grid Explorer **Questionnaire** to record your feedback on.

The study

The study is a series of steps to work through to give you a demonstration of the Grid Explorer. Your feedback is then captured on the accompanying feedback sheet.

Grid Explorer: an introduction

In this section we will introduce you to the basic initial steps that need to be taken to submit a source code

1. The Grid Explorer is already **launched**
2. Log on to the Grid by using password '**abdulrazaq**'
3. Submit your source code Put in the four information requested by the dialog box in the following order. For the **source code** put `\home\agah\agah.cpp`. For the input file type in **input.txt**. For the name of compiler drop down menu select **gcc**. The O/S Needed drop down menu choose **Linux**. Then click ok
4. You then see the progress of the source code compilation process.
5. When the source code is compiled it appears on the **Available Grids**, window.
6. You then need to drag the input file in `/home/agah/documents` onto the Executable
7. The result output.txt is then created.

Grid Explorer: the user interface

In this section familiarise yourself with the user interface and features

8. What can you say about the following features:
 - a. When you see the interface, do you immediately have an idea what it does for you?
 - b. The buttons in the Services/Application window, are they self explanatory
 - c. Are there other features that can be added to the interface to make it more intuitive or simple?

End of study: please complete the Questionnaire sheet – Thank you

Appendix F: Questionnaire Sheet.

Questionnaire for what Users think of the Grid Explorer User interface for Job Submission

Please answer this question with reference to the task that you have just performed.

Evaluators name:

Department/Group you belong to:

Are you: Male Female

Indicate your agreement or disagreement with the following statements

1. The startup user interface depicts what the system wants to achieve
 - Strongly disagree
 - Disagree
 - Neither agree or disagree
 - Agree
 - Strongly agree

Comments/Suggestions

2. It is easy to recover from mistakes
 - Strongly disagree
 - Disagree
 - Neither agree or disagree
 - Agree
 - Strongly agree

Comments/Suggestions

3. It is easy to get help when needed
- Strongly disagree
 - Disagree
 - Neither agree or disagree
 - Agree
 - Strongly agree

Comments/Suggestions

4. I always know what the system is doing with appropriate feedback
- Strongly disagree
 - Disagree
 - Neither agree or disagree
 - Agree
 - Strongly agree

Comments/Suggestions

5. I always know how well am doing
- Strongly disagree
 - Disagree
 - Neither agree or disagree
 - Agree
 - Strongly agree

Comments/Suggestions

6. The system tells me what to do at every point

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

Comments/Suggestions

7. The interface was easy to use

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

Comments/Suggestions

8. The dialogs and menus are easy to understand and consistent through out the interface

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

Comments/Suggestions

9. The errors messages I see are clear and precise to me.

- Strongly disagree
- Disagree
- Neither agree or disagree
- Agree
- Strongly agree

Comments/Suggestions

Definitions

Easy to use

For the context of this report is software environment that is simple to install and meets all the general principles (Heuristics) for a good user interface design

Stakeholders

The scientist in different professions who would like to be ordinary users of the Grid

Grid Middleware

The set of tools needed to access and make use of resources on the Grid.

GRIS/GIIS

Grid Resource Information System is an information provider component. Grid index Information Service is the configurable aggregate component. They are the globus Grid architecture high level services that allow Storage Request Brokers access data that can be used on an application level.

Grid Network

The collection of resources both software and hardware, responsible for the processing and execution of jobs, data management and job monitoring in a networked environment.

Glossary

ACAG-	Advanced Computer Architecture Group
ATM-	Asynchronous Transfer Mode
AURA-	Advanced Uncertain Reasoning Architecture
BPEL4WS-	Business Process Execution Language for Web Services
CA-	Certificate Authority
CAD-	Computer Aided Design
CARMEN-	Code Analysis, Repository, and Modeling for e-Neuroscience
CFD-	Computational Fluid Dynamics
COG-	Commodity Grids
DAI-	Data Access Integration
DAME-	Distributed Aircraft Maintenance Environment
EDG-	European Data Grid
ELF-	Executable and linking Format
GCC-	GNU Compiler Collection
GIIS-	Grid Index Information Service
GIS-	Grid Information Service
GRIS-	Grid Resource Information Service
GSI-	Grid Security Infrastructure
GT-	Globus Toolkit
GUI-	Graphical User Interface
HCI-	Human Computer Interaction
IDE-	Integrated Development Environment
I-WAY-	Information Wide Area Year
JVM-	Java Virtual Machine
LSF-	Load Sharing Facility
MCS-	My Condor Submit
OGSA-	Open Grid Services Architecture
OGSI-	Open Grid Services Infrastructure
PBS-	Portable Batch System
SDE-	Signal Data Explorer
SOAP-	Simple Object Access Protocol
WRG-	White Rose Grid
WSDL-	Web Services Description Language
XML-	Extensible Markup Language

References

Book

- [1.] Alhir, Sinan Si. *UML in a Nutshell*. O'Reilly & Associates, Inc CA, 1998
- [2.] Bennett, J., Holtzblatt, K. and Whiteside, J. *Usability engineering: Our experience and evolution*. In M. Helander, editor, Handbook of Human-Computer interaction. North-Holland, Amsterdam, 1988.
- [3.] Card, S.K., Robertson, G.G. and Newell, A. *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.
- [4.] Comer, Douglas E. *Essentials of Computer Architecture*. Pearson Prentice Hall, New Jersey, 2004.
- [5.] Crichlow, Joel M. *The essence of distributed systems*. Prentice Hall, 2000.
- [6.] Dix, Alan.J., Beale, Russell., Finlay, E. Janet., and Gregory, D. Abowd. *Human-Computer interaction*. 2nd Edition. Prentice Hall Europe, 1997.
- [7.] Dix, Alan.J., Beale, Russell., Finlay, E. Janet. and Gregory, D. Abowd. *Human-Computer interaction*. 3rd Edition. Prentice Hall Europe, 2003.
- [8.] Dumas, J. S. & Redish, J. C., *A Practical Guide to Usability Testing*. Norwood, Ablex Publishing, 1993.
- [9.] Foster, I., Kesselman, C. *The GRID: Blueprint for a New Computing Infrastructure*. 1999. San Francisco, California: Morgan Kaufmann Publishers, Inc.
- [10.] Faulkner, X. *Usability Engineering*. London, MacMillan Press Ltd, 2000.
- [11.] Graham Steve et al. *Building Web Services with java: Making sense of XML, SOAP, WSDL and UDDI*. SAMS PUBLISHING, 2001
- [12.] Nielsen, J. *Usability Engineering*. London, Academic Press Inc, 1993.
- [13.] Nielsen, J., Mack, Robert.L. *Usability Inspection Methods*. John Wiley and sons, Inc, New York. 1994.
- [14.] Rubin, J., *Handbook of Usability Testing: How to Plan, Design and Conduct Effective Tests*. New York, John Wiley & Sons, Inc, 1994.
- [15.] Shackel, B. & Richardson, S., *Human Factors for Informatics Usability*, Cambridge University Press, 1991.

- [16.] Sommerville, I. *Software Engineering*. 6th Ed. Addison-Wesley Publishers Limited, 2001.

Journal Article

- [17.] Allen, Gabrielle., Goodale, Tom., Davis, Kelly., Radke, Thomas., Russel, Michael and Seidel, Ed. *Application on the Grid: A GridLab Overview*. International Journal of High Performance Computing Application: Special issue on Grid Computing: Infrastructure and Applications. August 2003.
- [18.] DeFanti, T., Foster, I., Papka, R. Stevens and Kuhfuss, T. Overview of the I-Way: Wide area visual supercomputing. *Int. J. Supercomputing. Appl.*, 10(2), pg 123-130. 1996.
- [19.] *Edinburgh Parallel Computer Centre News*, The Grid-A new way to connect. Issue40, September 2000, pages 8-9.
- [20.] Folstad, Asbjorn. *Work-domain experts as evaluators: usability inspection of domain-specific work-support systems*. International Journal of Human-Computer Interaction 22 (3), pg 217-245, 2007.
- [21.] Foster, I. Kesselman, C. *Globus: A metacomputing infrastructure toolkit*. *Int. J. Supercomputing. Appl.* 11(2) pg 115-128, 1997.
- [22.] Foster, I., Kesselman, C., Tuecke, S. *The Anatomy of the Grid: Enabling Scalable Virtual Organisations*. International J. Supercomputer applications, 15(3), 2001.
Available From: <http://www.globus.org/research/papers.html#anatomy>
- [23.] Foster, I. *Globus Toolkit Version 4: Software for Service-Oriented Systems*. IFIP International Conference on Network and Parallel Computing, Springer-Verlag. LNCS 3779, pp2-13, 2006.
- [24.] Foster, I., Karonis, T. Nicholas., Toonen, Brian. *MPICH-G2: A Grid-enable implementation of the Message Passing Interface*. Journal of parallel and distributed computing. Vol 63, issue 5, Pages 551-563. May 2003
- [25.] Millikin, M.D. *DCE: Building the distributed future*, *Byte*, 19(June), 125-134. 1994.
- [26.] Mohan, C., Alonso, G., Gunthor, R., Kamath, M. *Exotica: A research perspective on Workflow Management Systems*. IEEE Data Engineering Bulletin, Vol. 18 No. 1, IEEE Computer Society, March 1995
- [27.] Pancake, C. M. *Usability issues in developing tools for the grid and how visual representations can help*. *Parallel Processing Letters*, 13(2), June 2003.

- [28.] Sjoberg, I.K Dag., Folstad, Asbjorn., Anda, C.D Bente. *The usability inspection performance of work-domain experts: An empirical study*. Interacting with Computers, Vol. 22 Issue 2, pg 75-87, March 2010.
- [29.] Taha, Y., Helal, A., Ahmed, K., Hammer, J. *Managing Multi-Task Systems Using Workflow*. In international journal of computers and Applications (IJCA), 21:3, pages 69-78, Sept. 1999.
- [30.] Vachhani, Milan K., Atkotiya, Kishor H. *Globus Toolkit 5 (GT5): Introduction of a tool to develop Grid Application and Middleware*. International Journal of Emerging Technology and Advanced Engineering
- [31.] Xu, F and Cox, S.J. *Workflow Tool for Engineers in a Grid-Enabled Design Search Toolkits*. Proceedings of UK e-science All Hands Meeting 2003, pp. 619-626

Proceedings articles

- [32.] Arrouvel, C., Austen, KF., Bruin, RP., Chiang, GT., Couch, PA., Dove, MT., Frame, I., Kleese Van Dam, K., Marmier, A., Murray-Rust, P., Parker, SC., Tyer, RP., Walker, AM and White, TOH. *Usable grid infrastructure: practical experiences from the eMinerals project*. Proceedings of UK e-science All Hands Meeting 2007.
- [33.] Attie, P., Singh, M., Rusinkiewicz, M. *Specifying and enforcing intertask dependencies*. In proceedings of the 19th VLDB conference, August 1993
- [34.] Bessis, Nik., French, Tim., Huang, Wei. and Maple, Carsten. *Can Intelligent Optimisation Techniques Improve Computing Job Scheduling In A Grid Environment? Review, Problem and Proposal*. Proceedings of Uk e-Science All Hands Meeting, 2006
- [35.] Austen, K. F., Blanchard, M.O., Bruin, R. P., Couch, P.A., Dove, M. T., Walker, A.M and White, T.O.H. *Job submission to Grid Computing environments..* Uk e-Science All Hands Meeting, 2006
- [36.] Brook, N., Harrison, K., Jones, R.W.L., Lavrijsen, W.T.L.P., Mato, P., Soroko, A., Tan, C.L and Tull, C.E. *GANGA: a user interface for Atlas and LHCb*. Proceedings of UK e-science All Hands Meeting 2003. ISBN 1-904425-11-9.
- [37.] Geddes, Neil., McAllister, Stephen and Richards Andrews. *Delivering support services for the National Grid servicet*. Proceedings of UK e-science All Hands Meeting 2007.
- [38.] He, Yunhui., Mo, Zan., Wu Guangwen and Wu Juhua.. Proceedings of the 2010 International *The Application of BP Neural Network in GridGain* Conference on Measuring Technology and Mechatronics Automation Volume 03. 2010.

- [39.] Jiang, Ping., Mair, Quentin., Newman, Julian. Using UML to design Distributed Collaborative workflows: From UML to XPDL. In Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies. June 09-11, 2003.
- [40.] Kandala , Savith., Sandhu, Ravi. *Secure role based workflow models*. In proceedings of the fifteenth annual working conference on database and application security. Pp 45-58, 2001
- [41.] Stevens, Robert. D., Robinson, Alan. J., Goble, Carole. A. *myGrid: personalised bioinformatics on the information grid*. In proceedings of 11th international conference on intelligent systems in Molecular Biology, 29th June-3rd July 2003, Brisbane, Australia, published Bioinformatics Vol. 19 Suppl. 1 2003, i302-304.
- [42.] Tang, J., Veijalainen, J. *Transaction-oriented Workflow Concepts in Inter-organizational Environments*. In proceeding of the 1995 ACM CIKM, pages 250-259, November 1995
- [43.] Majithia, Shalil., Taylor, Ian., Shields, Matthew., Wang Ian. *Triana as a Graphical Web Services Composition Toolkit*. Proceedings of Uk e-Science All Hands Meeting, 2003
- [44.] Rengger, R. *Indicators of usability based on performance*. In Proceedings of the Fourth International Conference on Human-Computer Interaction, Congress II: Design and Implementation of Interactive Systems: USABILITY EVALUATION; Development of Usability Metrics, vol 1, pp. 656-660, 1991
- [45.] Suciu, A.; Potolea, R.; , "A taxonomy for grid applications," Automation, Quality and Testing, Robotics, 2008. AQTR 2008. IEEE International Conference on , vol.3, no., pp.365-368, 22-25 May 2008

Report

- [46.] Aiken, R., Carey, M., Carpenter, B., Foster, I., Lynch, C., Mambretti, J., Moore, R, Strassner, J. and Teitelbaum, B. *Network Policy and Services: A Report of a Workshop on Middleware*, IETF, RFC 2768, 2000.
Available from: <http://www.ietf.org/rfc/rfc2768.txt>
- [47.] Buyya, R., Madiminti, K. Enterprise grid computing: State-of-the-art. Technical Report, GRIDS-TR-2005-16, Grid Computing and Distributed Systems Laboratory, The University of Melbourne
- [48.] Chin Jonathan, Coveney, Peter V. *Towards tractable toolkits for the Grid: a plea for lightweight, usable middleware*. Tech Rep UKeS-2004-01, UK e-Science. http://www.nesc.ac.uk/technical_papers/UKeS-2004-01.pdf, 2004
- [49.] Harrison, Tom. *White Rose Digital Factory: Project Report*. March 2004

- [50.] Workflow reference model. Technical report, Workflow Management Coalition, Brussels, 1994

Electronic Media

- [51.] A Computer Science Tapestry: Compiling, Projects, Libraries
Available from:
<http://www.cs.duke.edu/~ola/book/compiling.html>
[Accessed 30 July 2008]
- [52.] Allan, R.J et al . *A Review of UK HEC Grid Infrastructure: State of the Art and Next Steps*. 2000
Available from: <http://www.ukhec.ac.uk/publications/reports/ukhec-grid.pdf>
- [53.] *Association for Computer Machinery*
Available from:
<http://www.acm.org>
[Accessed 30 May 2004]
- [54.] *Aura Technology*
Available from:
<http://www.cs.york.ac.uk/arch/neural-networks/technologies/aura/background>
[Accessed 29 Nov 2009]
- [55.] *Carmen Project*
Available from:
<http://www.carmen.org.uk>
[Accessed 30 May 2004] pg10
- [56.] Cloud and Grid Computing
<http://www.brighthub.com/environment/green-computing/articles/68785.aspx>
[Accessed 13 May 2010]
- [57.] Cluster and Grid Computing:
http://www.digipede.net/downloads/Digipede_CCS_Whitepaper.pdf
[Accessed 3 January 2011]
- [58.] *Common Object Resource Broker Architecture*
Available from:
www.corba.org
[Accessed 30 Nov 2009]
- [59.] *Correlation Matrix Memory*
Available from:
<http://www.cs.york.ac.uk/arch/neural-networks/technologies/aura/background>
[Accessed 27March 2008]

- [60.] *Dame Project*
Available from:
<http://www.cs.york.ac.uk/dame>
[Accessed 27 March 2008]
- [61.] Distributed resource management application API (drmaa)
www.drmaa.org/wiki
[Accessed 10 Jan 2007]
- [62.] eMinerals Projects.
<http://www.eminerals.org/tools/mcs.html>
- [63.] *E-Legion*
Available from:
legion.virginia.edu/overview.html
[Accessed 30 May 2004]
- [64.] European Data Grid.
<http://www.eu-datagrid.org/>
- [65.] Foster I. *What is the Grid? A three point checklist*. July 20 2002.
Available from: <http://dlib.cs.odu.edu/WhatIsTheGrid.pdf>
[Accessed 08 December 2010]
- [66.] Foster, I. Internet Computing and the Emerging Grid. *Nature Web Matters*. 2000
Available from: <http://www.nature.com/webmatters/grid/grid.html>
[Accessed 15 February 2002]
- [67.] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Draft Document, <http://www.globus.org/research/papers/ogsa.pdf>, June 2002.
- [68.] Foster et al. <http://www.globus.org/ogsa>
- [69.] *Globus Toolkit*
Available from:
www.globus.org/toolkit
[Accessed 30 May 2004]
- [70.] GridGain White Paper
http://www.gridgain.com/media/gridgain_white_paper.pdf
- [71.] *IBM Web Site*
Available from:
(<http://www4.ibm.com/software/solutions/Webservices/pdf/WSCA.pdf>)
[Accessed 30 May 2004]

- [72.] International Standards Organisation
<http://www.iso.org>
- [73.] *Jini*
Available from:
www.sun.com/jini
[Accessed 30 May 2004]
- [74.] Linux Journal
Available from:
www.linuxjournal.com
[Accessed 30 July 2008]
- [75.] Load Sharing Facility
www.platform.com
[Accessed 10 February 2007]
- [76.] Model-View-Controller
<http://msdn.microsoft.com/en-us/library>
- [77.] *Mygrid Project*
Available from:
<http://www.mrgrid.org.uk>
- [78.] National eScience Centre
<http://www.nesc.ac.uk>
[Accessed 10 Jan 2008]
- [79.] Nielsens Usability Heuristics
http://www.useit.com/papers/heuristic/heuristic_list.html
[Accessed 10 Jan 2007]
- [80.] *Object Management Group*
Available from:
www.omg.org
[Accessed 30 Nov 2009]
- [81.] Open Grid Forum
<http://www.ogf.org/documents/GFD.44.pdf>
[Accessed 30 November 2010]
- [82.] Open Grid Forum
<http://www.gridforum.org/documents/GFD.113.pdf>
[Accessed 30 December 2010] pg9
- [83.] *Open Middleware Infrastructure Institute*
Available from:
<http://www.omii.ac.uk>
[Accessed 30 Jan 2007]

- [84.] Open Grid Services Architecture-Data Access Integration
Available from:
(<http://www.ogsa-dai.org.uk>)
[Accessed 30 July 2004]
- [85.] *Open Grid Services Architecture*
Available from:
<http://www.globus.org/ogsa>
[Accessed 30 May 2009]
- [86.] *Open Grid Services Infrastructure*
Available from:
<http://forge.gridforum.org/projects/ggf-editor/document/draft-ogsi-service-1/en/1>.
[Accessed 30 May 2004]
- [87.] Portable Batch Systems
www.nas.nasa.gov/NAS/projects/pbs
[Accessed 10 Jan 2007]
- [88.] Pinelle, D. & Gutwin, C. *Group Task Analysis for Groupware Usability Evaluations*. 2001.
Available from www.computer.org,
[Accessed 11 May 2002].
- [89.] San Diego Super Computing Centre
Available from:<http://www.sdsc.edu/srb/>
[Accessed 21 April 2008]
- [90.] *Service Oriented Computing*
Available from:
(<http://www.ebpml.org>)
[Accessed 30 Nov 2009]
- [91.] *Storage Resource Broker*
Available from:
http://www.sdsc.edu/srb/index.php/Main_Page
[Accessed 29 Nov 2009]
- [92.] Styx Grid Services
Available from:<http://www.resc.rdg.ac.uk/jstyx/sgs>
[Accessed 21 April 2008]
- [93.] *The eMineral Project*
Available from:
www.eminerals.org
[Accessed 28 March 2008]pg10

- [94.] *The Globus Toolkit*
Available from:
www.globus.org/toolkit
[Accessed 30 May 2004]
- [95.] *The Microsoft Library*
Available from:
<http://msdn.microsoft.com/library/default.asp>
[Accessed 30 May 2004]
- [96.] *The White Rose Grid*
Available from:
<http://www.wrgrid.ac.uk>
[Accessed 27 March 2008]
- [97.] *Triana Project*
Available from:
<http://www.trianacode.org> pg10
[Accessed 30 May 2004]
- [98.] Tuecke, S et al., *Open Grid Services Infrastructure (OGSI) –Version 1.0. 2003*: <http://forge.gridforum.org/projects/ogsi-wg>
- [99.] *UDDI Technical white paper*, September 6, 2002
Available from:
http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
[Accessed 30 May 2004]
- [100.] *Universal Description, Discovery and Integration*
Available from:<http://uddi.xml.org/>
[Accessed 27 March 2008]
- [101.] Unix Help Pages
Available from:
<http://unixhelp.ed.ac.uk/CGI/man-cgi?ld.so+8>
[Accessed 30 July 2008]
- [102.] *Usability.Gov (Guide for developing useful websites)*
Available from:<http://www.usability.gov/basics/index.html>
[Accessed 30 April 2010]
- [103.] Use Case Explained in Wikipedia
http://en.wikipedia.org/wiki/Use_case
[Accessed 15 March 2010]
- [104.] Ustun Yildiz, Adnene Guabtni, Anne H.H. Ngu: *Business versus Scientific Workflow: A Comparative Study; May 2009*
URL:<http://www.cs.ucdavis.edu/research/tech-reports/2009/CSE-2009-.pdf>
[Accessed 31 Nov 2010]

- [105.] *Webopedia*
Available from:
www.webopedia.com
[Accessed 30 May 2004]
- [106.] *Web Services Description Language (WSDL)*:
<http://www.w3.org/2002/ws/desc>
- [107.] *Web Services Resource Framework: The Globus Alliance*
http://www.globus.org/wsrp/specs/ogsi_to_wsrp_1.0.pdf
[Accessed 4 December 2006]
- [108.] *Workflow Definition*
Available from:
<http://searchcio.techtarget.com>
[Accessed 30 May 2004]
- [109.] *World Wide Web Consortium*
Available from:
(<http://www.w3.org/TR/wsdl>)
[Accessed 30 May 2004]
- [110.] *World Wide Web Consortium*
<http://www.w3.org/2001/03/wsws-popa/paper51>
[Accessed 30 May 2004]

Lecture Note

- [111.] Matravers, Julika. *Business Information Systems lecture handouts*, School of Computing, University of Leeds, 2001