GridAssist Middleware Interoperability
Through Scenario Based Software Architecture Assessment

P.N.J. Bouwmeester
September 4, 2005
GridAssist Middleware
Interoperability

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

SOFTWARE ENGINEERING

by

P.N.J. Bouwmeester
born in Alkemade, the Netherlands
© 2005 P.N.J. Bouwmeester. Document generated with \LaTeX, MiKTeX and WinEdt.

Cover picture: A representation of a global Grid, which is the goal of the grid community.
GridAssist Middleware
Interoperability

Author: P.N.J. Bouwmeester
Student id: 1064770
Email: p.n.j.bouwmeester@student.tudelft.nl

Abstract

This thesis describes the work done on adapting a grid computing client application. This tool, called GridAssist, was extended with a middleware framework to support multiple grid middleware environments. This framework can easily switch between protocols and services in order to allow simultaneous execution on resources that differ in grid middleware environment. With this framework it is possible to reach a larger amount of grid resources. In addition, the adaptations were investigated with software architecture assessment methods, in order to assure that the quality requirements are still met. This assessment focusses particularly on modifiability, portability and usability, which should not degrade by the design adaptations. Finally, a prototype of GridAssist that is compatible with the second and the fourth version of the Globus toolkit was delivered.

Thesis Committee:

Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: Dr. Ing. L. Moonen, Faculty EEMCS, TU Delft
Company supervisor: Drs. M. ter Linden, AS&E, Dutch Space
Committee Member: Prof. Dr. M. Neerincx, Faculty MMI, TU Delft
Simplicity is the ultimate sophistication
- Leonardo da Vinci
The grid connects various resources, ranging from sensors to desktop computers, in such a way that it provides dependable, consistent and secure access for its users. A grid can be used to increase the capacity of data storage or computing power. In the optimal situation the grid should become for computing power what the web is now for information. For this goal it is necessary that all the connected resources and middleware software can be communicated with. Until recently, standardization for grid technology was limited, causing many different incompatible grid solutions to be implemented. The interoperable connection of various separated devices in a uniform way gets compromised, which resulted in many independent grid middleware environments that cannot communicate with each other. Fortunately the Global Grid Forum is providing the grid community with standard protocols and services, in order to bring the grid technologies together and realize interoperability.

This project focuses on a grid client application that allows job submission and data transferring for a grid. This application, called GridAssist, is used by data processing scientists to submit jobs to a grid. GridAssist has been extended and adapted to allow communication with multiple grid initiatives. The goal of this project is to make it possible for GridAssist to reach all assets of a divided community and provide GridAssist’s users with a more diverse pool of resources. This was done by adding GridAssist with a grid middleware interoperability framework that allows communication with multiple grid middleware environments. The design process has been performed according to the software design standard of the European Space Agency.

The developed framework provides the possibility for multiple wrappers to be plugged into GridAssist, each wrapper providing communication to one grid middleware environment. A research to the most promising grid middleware environments has shown that Globus Toolkit version 2, Globus Toolkit version 4, gLite and Uniform
Interface to Computing Resources (UNICORE) are currently the best candidates for integration with GridAssist. A wrapper for the Globus Toolkit version 4 middleware environment has been implemented and tested. And a wrapper for gLite has been researched but could not be completed with the current version of this middleware. This framework should allow easy adoption of future developments of grid middleware environments.

Because changes to GridAssist’s software architecture should not compromise any previously defined quality requirements, a software architecture assessment was performed to guarantee that GridAssist maintains the same level of quality in respect to modifiability, usability and portability. Three methods were used to assess these attributes and it was shown that neither modifiability and portability were negatively influenced by the design adaptations. However, the architectural usability support was shown to differ from the requirements. The architecture’s focus on (usability sub-attributes) efficiency and reliability is good, but should focus more on satisfaction and learnability. Furthermore, a new assessment method was used to investigate the screen complexity difference due to the adaptations to GridAssist’s architecture. This indicated a relatively minor increment in the number of elements (windows, buttons, input fields) per scenario, but with a wide spread: thirty nine percent of the scenarios are affected by the changes. Fortunately, each of these scenarios have only a minor complexity increase.

A user walkthrough has shown that the changes to GridAssist do not significantly influence the way users experience the complexity of GridAssist. This walkthrough was performed to compare the results of the usability assessments with real-life experience of users. The walkthrough was performed after the implementation and testing phase.

Because the Global Grid Forum is developing standards for all facets of the grid, the technologies are coming closer and closer together. This makes it easier and easier to combine the grid middleware environments. This project has shown that an interoperability layer can realize interoperability in a basic way. Also, this project has shown that scenario based assessment is a versatile method that, because it is customized for each setup, allows all kinds of assessments of software architectures.
Acknowledgement

This report is the result of a graduation project in completion of the Master’s program Computer Science and is supported by the Software Evolution and Research Laboratory (SWERL) at the University of Technology in Delft. This project was carried out between February and August 2005 at Dutch Space B.V. in Leiden. It has been an interesting and challenging project, which would never have been possible without my two supervisors: Mark ter Linden and Leon Moonen. They provided me with feedback, starting points for research and answers to my questions. Therefore, they both deserve much gratitude.

I would also like to thank my colleagues at DutchSpace, but especially Jan Kokenberg for reviewing my thesis and for being a cheerful colleague during my internship, Hans de Wolf for providing the thorough review of my thesis which was exactly what I needed, and also Erik Laan, Leendert Vijfwinkel and Aad Eegers for their cooperation in the walkthrough. And furthermore, I would like to thank Mark Neerinxc (part-time professor at the TUDelft) for putting time into a discussion about usability. His insightful comments inspired me to further investigate the usability aspect of my software design.

Much gratitude go to my parents, who continuously supported me with everything they got, and who are the foundation (and fund-ation) for every accomplished dream. Also many thanks go to Sylvia, who was always there to listen to my complaints, problems, complex stories and happy endings. And last, many thanks go out to all the people who helped, assisted and stood by me during my study, my internship and during the other daily things I did the last five years.

Kevin Bouwmeester
Leiden, the Netherlands
August 12, 2005
## Contents

Management Summary ........................................ v
Acknowledgement ........................................ vii

1 Introduction ........................................... 1
  1.1 Dutch Space ........................................ 2
  1.2 Problem Statement ................................ 3
  1.3 Project Objectives ................................ 4
  1.4 Approach ........................................... 5
  1.5 Outline ............................................. 5

2 Grid Computing .......................................... 7
  2.1 Why a Grid? ........................................ 8
  2.2 Virtual Organizations .............................. 9
  2.3 Grid Types ......................................... 10
  2.4 Advantages of Grids ............................... 11
  2.5 Grid Applications ................................ 11
  2.6 Grid Standardization ............................. 12
  2.7 Grid Middleware .................................. 16

3 GridAssist ............................................... 25
  3.1 The Flow of Work ................................ 25
  3.2 Design for Assistance ............................. 27
  3.3 Communication ................................... 29
  3.4 Further Evolution of GridAssist ................. 29
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Designing Interoperability for GridAssist</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Requirements Specification</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Architectural Design Solutions</td>
<td>34</td>
</tr>
<tr>
<td>4.3</td>
<td>Architectural Design Details</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Further Design Steps</td>
<td>40</td>
</tr>
<tr>
<td>4.5</td>
<td>Testing</td>
<td>41</td>
</tr>
<tr>
<td>4.6</td>
<td>The good, the bad and the ugly</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Architecture Assessment Methods</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>Existing Methods</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>An Additional Assessment Method</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>Architecture Assessment Process</td>
<td>59</td>
</tr>
<tr>
<td>6.1</td>
<td>Method Applicability</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Assessment Setup</td>
<td>62</td>
</tr>
<tr>
<td>6.3</td>
<td>Modifiability Assessment</td>
<td>63</td>
</tr>
<tr>
<td>6.4</td>
<td>Portability Assessment</td>
<td>64</td>
</tr>
<tr>
<td>6.5</td>
<td>Screen Complexity Assessment</td>
<td>66</td>
</tr>
<tr>
<td>6.6</td>
<td>Usability Assessment</td>
<td>69</td>
</tr>
<tr>
<td>6.7</td>
<td>Conclusions</td>
<td>76</td>
</tr>
<tr>
<td>7</td>
<td>Related Work</td>
<td>81</td>
</tr>
<tr>
<td>7.1</td>
<td>Grid Computing</td>
<td>81</td>
</tr>
<tr>
<td>7.2</td>
<td>Software Architecture Assessment</td>
<td>82</td>
</tr>
<tr>
<td>8</td>
<td>Concluding Remarks</td>
<td>85</td>
</tr>
<tr>
<td>8.1</td>
<td>Discussion</td>
<td>85</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Work</td>
<td>87</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>89</td>
</tr>
<tr>
<td>A</td>
<td>Acronyms</td>
<td>95</td>
</tr>
<tr>
<td>B</td>
<td>Requirements on Design</td>
<td>99</td>
</tr>
<tr>
<td>C</td>
<td>Globus Toolkit 4 Wrapper Source Code</td>
<td>105</td>
</tr>
<tr>
<td>D</td>
<td>User Walkthrough</td>
<td>113</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The trend in both commercial and technical products is Internet-connectivity, more and more devices are getting ’wired’ to the Internet. Sensors, super-computers and data storage resources are but a small example of the connected systems. Because so many different tools and systems are reachable over the Internet, it becomes possible to implement a structured and controlled way of using Internet-connected hard- and software.

That is where a grid comes in; it allows dependable, consistent, pervasive and inexpensive access to resources. These resources can range from super computing clusters to Radio Frequency Identification (RFID) sensors. The term grid comes from the electric power grid, which can be seen as an example for the computing grid. Electric power grids provide cheap and reliable power in such a way that both individuals and industries take its availability for granted. This is where the computational grid is headed, although there are still many barriers to overcome.

The infrastructure of computational grids is based on the large-scale pooling of resources (sensors, people, computing cycles, or data). Grids should provide a dependable service to its users, who will expect reliable, predictable service with a high performance [28].

Another attribute of grids, just as in the electrical power grid, is the need for standard services, accessible trough a general interface. For example, users of the power grid also demand standardized access to the power connectors. The difficulty in standardization for the computational grid, lays in weighing performance and heterogeneity. A standard interface sometimes compromises the performance characteristics of the application. Furthermore, without the constraints of standardization an application can be tuned towards one specific goal, increasing its performance.
Currently, performance is favored and heterogeneity is undermined by a great variety of middleware software. This middleware should function as the layer between the user and the resources that form a grid. There are multiple popular implementations of this middleware software. Some were developed for a particular grid project (e.g. the Large Hadron Collider Computing Grid), while others are delivered as general-use packages (e.g. Globus Toolkit). Some are produced by the open-source community and others are commercial software packages. Due to all the different ideas on how to build a grid and on how to connect all the diverse resources, one single interoperable grid (like the Internet) is still far away.

Fortunately, there is an initiative to unite grid developments and bring the grid community towards a more standardized cooperating field. The Global Grid Forum (GGF) is developing standards for grid technology, it is getting more and more effective and influential as more and more parties get involved in it.

1.1 Dutch Space

In the sixties, the airplane construction company Fokker started a department for space products. Within twenty years, this department grew immensely and in 1987 it became independent. Formally it was still a daughter-company of Fokker under the name Fokker Space, but after Fokker’s bankruptcy it became autonomous and its name was changed into Dutch Space. Today Dutch Space is Europe’s leading manufacturer of solar arrays, and is also involved in the development and production of parts of the Ariane rocket. Furthermore, they are the leading producer of the European Robot Arm for the International Space Station, which will be attached to the Russian part of the station in 2007.

The work described in this thesis has been done for the data processing department in the Advanced Systems and Engineering division. Data processing is concerned with processing large amount of data in order to generate useful information. Some customers of Dutch Space process data that comes in from environmental satellites towards more readable, understandable and interpretive information of interest. One example is data that is gathered by the Ozone Monitoring Instrument (OMI) that is on board of the AURA satellite that was launched by NASA in July 2004. Dutch Space works in close collaboration with parties such as European Space Agency (ESA) to develop technological solutions that allow scientists to easily use the most advanced technologies for data processing tasks.

One of these solutions is a tool that supports scientists in accessing data- and computing grids through a graphical user interface. This tool, called GridAssist,
has been developed from a usability perspective and completely hides technological implications concerning computing grids. GridAssist’s objective is to provide support for the creation, submission and monitoring of workflows. A workflow consists of several computational applications and storage nodes, where the output of one application can be directly used as input for several other applications. When the former application produces multiple output files, each of them can be used as input for a separate application following in the workflow. The workflow that is used for processing the raw data that comes from OMI, includes steps for filtering the data, correcting for any errors, a pre-processing step, a data analysis step and finally the generation of a picture that shows the ozone layer thickness. A recently generated image can be seen in figure 1.1. GridAssist allow users to submit workflows to a grid and, when processing is finished, the results can either be distributed over storage resources, or downloaded to the end-user’s computer.

GridAssist can be seen as the user’s viewing window to grid middleware. This window gives the user an understandable view of, and insightful access to the basic functionality that grid middleware offers. Customers of DutchSpace that use GridAssist are for example ESA and KNMI.

1.2 Problem Statement

The grid community is built of many different middleware implementations, but GridAssist currently only supports one of them. In the most ideal situation, all the separated grids will merge to one great, interoperable grid. However, this utopia will not be realized in the near future, because there are simply too many parties, and besides that, a strong authoritative agency is missing. In order to reach more customers and to run applications on more grid nodes, GridAssist needs to be interoperable with at least the most promising grid environments. It shall be advantageous for both Dutch Space and its customers when GridAssist supports more middleware implementations.

Removing the current restriction of one grid middleware environment has both commercial and technical implications. First, if more grid middleware environments are supported, a greater diversity in reachable grid resources can be obtained. And more grid nodes can provide more computational power, which means better performance of execution of the workflow. GridAssist’s currently supported middleware environment is still the major one in use, but is slowly becoming outdated. Future grid resources will probably run on newer grid middleware, so wider middleware support safeguards future operation of GridAssist. Furthermore, because there are more resources to execute on, the user has a wider choice for the best performing resources.
Also, interoperability between various grids will greatly enhance the possibilities of grid computing, as more computing power and more storage space will be available. And finally, when more grid resources are supported, a wider customer area will unfold.

1.3 Project Objectives

The main objective of this project was to extend GridAssist with a framework that supported multiple middleware environments. Besides the software design adaptations, there will also be a particular focus on how to perform the transformation of the software architecture of GridAssist. How to make adaptations in such a way that no decay of the architecture occurs? When the addition of functionality has negative consequences on software quality, this is referred to by software architecture decay [22].

Figure 1.1 — Picture generated from OMI data
1.4 Approach

To be able to decide on the most promising grid middleware environments, this thesis starts with an investigation of current grid middleware implementation differences. Subjects that are discussed range from components and protocols to usage and standardization. With this information the current software architecture, software requirements and design can be closely studied. Based on the current requirements, the objectives and the results of the research the requirements specification phase will define the constraints and qualities for the remainder of the project.

Next, several proposals for adapting the current software architecture were created. The ideal solution, according to the software requirements is worked out in an architectural design. This architecture is analyzed by a scenario-based assessment [34], which will make sure that the architectural qualities stay within specifications.

Eventually, a prototype of the adapted GridAssist software will be delivered, based on both the design for interoperability between grid middleware and the scenario based assessment. From hereon, the interoperable version of GridAssist will be referred to as adapted GridAssist.

1.5 Outline

This thesis can be viewed as if divided into three parts that follow the approach as described above. It might be worth to note that an explanation of acronyms is given in appendix A.

Part I provides preliminary information and presents the results of the research phase. Chapter 2 discusses general issues concerning grid computing, the standardization initiatives and the grid middleware solutions that are relevant for Dutch Space. In relation to this information, chapter 3 discusses the objectives and technologies of GridAssist in more detail.

Part II focuses on the requirements analysis and design phases of the interoperability aspects of GridAssist. Chapter 4 describes the user and the software requirements on the adaptations that were done to GridAssist. This chapter also gives a description of the design and test activities. Finally, this chapter concludes with a reflection on the design.

Part III deals with the evaluation of the software architecture according to the quality requirements. The architectural design which was described in the second part is used as input for this assessment. Chapter 5 describes several existing meth-
ods and also presents a new method. In chapter 6 three methods are selected to be used in the assessment of GridAssist’s software architecture. Furthermore, the results of the assessment are presented and discussed.

Chapter 7 lists important literature and work that is related to this project. Finally, this thesis concludes with chapter 8 giving a conclusion on the whole project and providing recommendations for follow-up projects.
Chapter 2

Grid Computing

The Internet has evolved from an information network for scientists into a widely used medium for sharing files and information. If we imagine how our modern society would look like without the availability of the Internet, we would awake in a really different world. Our current rate of communication and information sharing would not be possible without the existence of this fast, reliable and interactive network. The Internet can be seen as an influential technological development, that can be used for sharing information and computing power. But why not take it one step further? Why not share data, sensors and computing power in a flexible, secure and transparent way?

As an example, imagine yourself making stock predictions for general use. First you access the stock monitoring resource that has the information needed for your prediction algorithm. After that you start your algorithm on a remote supercomputer that meets your algorithm’s specifications. A super computer can perform the Monte Carlo\(^1\) analysis within fractions of the time it would take on your home computer. And after you have obtained the results you can easily distribute the predictions over several data storage resources, so that people can easily access this information. And all this without you ever caring where the resources are located. This scenario can be realized in a grid, which makes computing power and storage space transparent over any computer network (e.g. the Internet).

The term grid was first used ten years ago and comes from an analogy with the power grid. Because the availability and easiness with which computing power is shared through a grid is comparative to the easiness with which electrical power is shared over the power grid. As is stated on gridcafe\(^2\): ”A grid would let users tap

---

\(^1\)Monte Carlo methods are stochastic techniques; they are based on the use of random numbers and probability statistics to investigate problems.

\(^2\)A web site offering an understandable view on a grid: \texttt{http://gridcafe.web.cern.ch}
processing power off the Internet as easily as electrical power can be drawn from a wall socket. Currently there are multiple grid initiatives, so we cannot speak of the Grid, as we can of the Internet. In this thesis I use a grid to indicate an example of a current grid and the Grid when I speak of a future global grid that can be compared to the Internet.

A general definition on what a grid exactly is is hard to give. Many parties have provided a definition according to their own perspective on grids. This makes a grid a vague concept, which does not contribute to the adoption of its technologies. To be able to distinguish a grid from a non-grid initiative, Ian Foster has constructed a three-point checklist [27]. According to this checklist a grid is a system that...

1. ...coordinates resources that are not subject to centralized control. The resources do not belong to the same control domain; it is a grid that integrates and coordinates the resources and users.

2. ...uses standard, open, general-purpose protocols and interfaces. The system should not be application specific and be multi-purpose in its protocols and interfaces.

3. ...delivers nontrivial qualities of service. The resources should together be able to provide a quality level that is greater than the sum of the parts. For example, provide better response time, throughput, availability or security, which makes the utility of a grid significantly greater.

Besides this checklist, it is important to indicate that a grid is not an unlimited resource for computing power. The availability of computers, support and bandwidth influences the performance of a grid. The more computing power you demand, the more hardware, software and support is needed. Also, setting up, using and maintaining a grid will cost a lot of money (depending on a grid’s size), as you still have to pay for computers, administration, bandwidth usage and maybe also for used clock cycles.

2.1 Why a Grid?

It is due to several developments and trends that the existence of grids is now a fact. First of all, there has been a great increase in computing power from the Cray supercomputer of ten years ago until the modern BlueGene\textsuperscript{3}. But the greatest increase in computing power, within a reasonable budget, can only be realized by the coupling of multiple (less powerful) computing resources. Even in desktop pc’s the

\textsuperscript{3}IBM’s supercomputer, holds top position on the top 500 supercomputer list. See www.research.ibm.com/bluegene
trend from one processor to dual cores, or multi-processor systems is noticeable. One of a grid’s purposes is to combine several computing resources, in order to obtain more computing power.

Secondly, there is a trend of demand-driven computations which have only episodic need for high computing power, for example medical imaging systems only need computations when a patient is being examined, and seismic simulations only when an earthquake has taken place. A grid could provide reliable, instantaneous, and transparent access to high-end resources. From the application’s point of view it is as if the resources are dedicated to them [28].

Furthermore, a grid can use otherwise idling computing resources (e.g. desktop pc’s and workstations). This way a grid provides great computing power of otherwise wasted clock cycles. And besides, a grid shares more than just computing power, also otherwise wasted storage space can be shared, so large results of complex calculations can be made available to multiple parties. An example is the weather forecast, which involves extensive numerical calculations and who’s output is distributed and used by many people.

2.2 Virtual Organizations

The Internet brings people from different companies, who can be located many kilometers from each other, together. An organization that is composed of multiple parties who are geographically distributed but who appear to the outside world as one organization with a real physical location, is called a Virtual Organization (VO). Individuals within a VO share resources in a highly controlled manner, with resource providers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. Virtual organizations are mostly project based and therefore temporarily. Examples of VO’s are: members of a large international multi-year high-energy physics collaboration, consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory, and Application Service Provider (ASP) [29].

Current distributed computing technologies do not address the concerns and requirements of a VO. A VO requires technologies that are highly flexible in sharing relationships, provide complex and high levels of control over how shared resources are used, share varied resources, and allow diverse usage modes (from single user to multi-user and from performance sensitive to cost-sensitive). Over the past five years grids have evolved to include protocols, services and tools that address exactly these requirements. A grid is especially suitable to support VO parties in using each
other’s resources and to share information.

Because a virtual organization shares all kinds of resources, security is very important. Without proper security secret business information will be open to all grid users. Participants of a virtual organization are willing to share information, data, sensors and computing power to others, but not everything to everybody. Therefore, users should be authenticated and authorized; not everyone should be able to access all the available information and resources (e.g. sensors, computing servers). The user-rights of employees of a VO should be carefully mapped onto authorization information of grid resources. For example, resources are mostly shared between users who belong to the same resource. Various grids have based their security measures on identifying people and the VO(‘s) they belong to.

2.3 Grid Types

Because grids are so versatile, they appear in many different forms and perform many different functions. Besides grid-based high-end super computers, there are examples of grids in which internet-connected desktop computers can participate. A normal desktop computer user utilizes only a small percentage of the computing power that his/her Personal Computer (PC) provides. Furthermore, a PC spends a considerable time in ‘screen-saver mode’, for example when the user is away for lunch. What about using all this unused computing power for more practical purposes than just displaying a screen saver? Two initiatives which are aimed at doing academic discoveries are Seti@home\(^4\) and Einstein@home\(^5\). The former is a well-known project that searches for signals of extraterrestrial life, and the latter is an attempt to use desktop computers to process measurement data in order to find prove for the existence of gravitational waves. These examples do not satisfy the three-point checklist, because a grid is not as restrictive in its protocols as the Seti@home and Eisntein@home. But they resemble two distributed computing examples that represent the forerunner of today’s desktop grids.

The type of grid that is used by GridAssist is the cluster grid, which consists of several clusters of dedicated computers. A cluster consists of several computers that act as one single high performance computer to the outside world. Computational jobs are split into separate tasks and divided among the processors in the cluster. An example of a cluster grid that is partly located at the university in Delft is the Second Distributed ASCI Supercomputer (DAS-2) which consists of 200 nodes grouped in 5 clusters.

\(^4\)http://www.setiathome.com
\(^5\)http://www.einsteinathome.com
Other grid types that are commonly encountered are High Performance Computing (HPC) grids, and data grids. The former specifically designed to perform complex calculations, and the latter for high speed storage and retrieval of large data sets. Grids are sometimes also classified according to the structure of the organization in which it is used: departmental, enterprise, extraprise and global grids.

2.4 Advantages of Grids

Besides the possibilities of a grid mentioned above, a computing grid has significant advantages for companies that heavily rely on computing power. The average utilization of all IT resources within a company is about 10% [31], which makes today’s information technology infrastructure rather inefficient. Grids do require some investments regarding deployment and installation, but resources will be used more efficient, which lowers required hardware investments and resources. A desktop grid can be deployed on all employees’ computers, which saves the costs of building and maintaining (electricity and air conditioning) a server room.

The operational expenses of a grid are much lower than that of normal HPC resources. Grids have the ability to cross organizational and regional borders, which uniformly increases the level of computational capacity, and enhances the level of redundancy in the infrastructure. Because the granularity of this redundancy is much larger than seen in HPC resources, it is a step forward against outages.

In the light of current computing power resources, an information manager is forced to increase the computing power in large steps (e.g. buy a new computing server), when the requirements slightly increase. A computing grid provides a scalable and flexible enterprise Information Technology (IT) infrastructure that accommodates ranging requirements in computing power. Resources can be added linearly based on real-time business requirements.

A grid’s throughput is limited by the speed of the network connections, but when these are fast enough a grid has higher performance (in respect to data throughput and computing power) in comparison to HPC resources. Its implementation can accelerate product development, improve time to market, and raise customer satisfaction [1].

2.5 Grid Applications

The applications that are to run on a grid can be divided into five classes, each having its own characteristics which make a grid extremely useful for them:
• Distributed computing involves very large problems that require lots of Central Processing Unit (CPU) cycles and memory, and thus can not be solved on a single system. A grid can provide the required amount of CPUs and memory. Examples can be found in the field of stellar dynamics, where computations on large quantities of data need to be performed.

• High throughput computing uses a grid to schedule a large number of small and independent tasks, in order to utilize otherwise unused processor cycles.

• On demand computing requires great amount of computing power only once in a while, and often on a short-term basis. A grid makes this computing power available as if it is dedicated to the on-demand application. Examples include medical instrumentation and cloud detection.

• Data intensive applications work on large data sets in order to obtain new information. A grid can be used to divide the enormous quantities of data and make calculations on the data possible. An example is the Large Hadron Collider (LHC) [45], which is a particle accelerator that is going to produce large quantities of data (operational in 2007). The measurements of particle collisions are divided among grid nodes, and calculations are performed on these separated parts of the data set. Furthermore, a grid acts as one single enormous storage device, for easy and fast data access a grid searches the appropriate data and returns it to the user.

• Collaborative applications are used to support communication and collaborative work between multiple participants. These applications are concerned with the real-time requirements imposed by human perceptual capabilities. A grid can help in meeting these requirements by providing high throughput and processing power.

These applications require many computational resources and rely on the availability of a grid infrastructure that guarantees robust performance. These requirements are met by the functionality and quality that a grid offers.

2.6 Grid Standardization

Many different technologies and protocols have been developed during the last ten years of grid development. But now, researchers are working to come to a more standardized way of connecting resources to a grid. This sometimes means choosing the best of all alternatives and sometimes means developing of a more suitable standard.
2.6.1 Global Grid Forum

What the Internet Engineering Task Force (IETF) is for the Internet community, the GGF is for the grid community. The structure of the GGF has been modeled after the IETF; it consists of research and working groups. Research groups have been created to closely examine some areas of grids and identify where standards are necessary. A group can spawn several working groups, each working on one facet of grid technology. Working groups are mostly concerned with the development of current implementations towards a standard. Participants gather at GGF meetings, which are organized several times a year all over the world. Unfortunately, the cooperation in GGF is purely on voluntary basis, which results in powerful parties dominating over the standardization process and other parties totally ignoring the GGF’s standards.

2.6.2 Between Grid Services and Web Services

The Open Grid Service Architecture (OGSA), was proposed by the GGF to become the standard architecture for grid applications [17]. This architecture provides a middleware framework that defines the requirements for grid services (job submission, data transfers, resource monitoring, etc). The logical middleware is defined in terms of services that are to be provided, the interfaces that these services expose, the individual and collective state of the resources belonging to the services and the interaction between these services. According to OGSA, each particular service should be accessible regardless of vendor, organization, and internal implementation. This framework has been implemented in the Open Grid Service Infrastructure (OGSI), which is a specification for grid services.

With OGSI, web services can be used to build a Service Oriented Architecture (SOA), which describes how several independent services cooperate to undertake a common task. The SOA improves the interoperability and reusability of web services. The GGF expected that the web service standards and the OGSI would convergence, but this was not happening. This was mainly caused by several objections about OGSI by the web service community [17];

- Firstly, the OGSI specification is complex and dense. There is no clean separation of functionality to support incremental adoption.
- Secondly, OGSI uses functions that do not work well with existing web services tooling. Because the XML specifications are not always followed by rule, problems can arise with tools that do expect exact conformance with XML specifications. For example, OGSI often uses of \( xsd : \) any, which can cause problems with for example JAX-RPC. Because this element is unrestricted,
any parameter type can be used in JAX-RPC, which can cause conversion problems [48].

- Furthermore, OGSI is too object-oriented. This is specifically a problem in the stateful services that OGSI defines. A stateful resource can be defined as follows: the resource is composed of a state, which is defined in XML format, the resource has a life cycle and the state of the resource can be manipulated by other web services [17]. A set of conventions and extensions on the use of Web Service Description Language (WSDL) and XML Schema enable stateful web services. But web services should not have an internal state and should compose their output from only their input. The identity and life cycle of the service and the resource state should not be coupled [17].

Unfortunately, the above objections stalled the standardization process that the GGF had in mind. Some additional measures were necessary to define a SOA for grids.

2.6.3 Web Service Resource Framework

In order to bridge the gap between popular web services and grid services of OGSI, the GGF has proposed a refactored version of OGSI [17, 18]. This refactoring is called the Web Service Resource Framework (WSRF), it should make resources accessible according to web service specifications. WSRF should be seen as a part of the web service standards, instead of a wrapper over existing standards like OGSI. But WSRF is no radical change, it is conceptually the same thing as OGSI, but now within the specifications of the web service community. Two web service standards are really important for WSRF:

**WS-Resource** is the combination of a web service and a resource that holds the internal state of the web service.

**WS-Addressing** specifies how a WS-Resources is addressed. This is done by end point references, that specifies both the web services and the resource to use.

Eventually, WSRF should become a replacement of the OGSI standard, thereby satisfying the web service community. The objections towards OGSI are not applicable for WSRF [17];

- WSRF has been specified in five documents plus a complementary specification on top of that. This separation of specification makes WSRF less complex [18].

- WSRF better exploits existing eXtensible Markup Language (XML) standards, which makes it easier to implement within existing web service toolkits.
- Furthermore, WSRF defines a way to allow stateful resource manipulation via web services operations. This is achieved by putting the state in a separate entity, the resource. Whenever a stateful interaction with a WSRF service is made, the caller simply has to instruct the service to use a particular resource. This way, the web service remains a simple message processor without a state, but when combined with a resource entity it can be used as a stateful resource.

Figure 2.1 — Convergence between grid and web

Figure 2.1 depicts this convergence of the web and grids, both starting with great difference in application and technology. These two technologies are slowly converging as new developments arrive. Along the grid line some major developments are given, starting Globus Toolkit version 1 (GT1) which was one of the first grid middleware implementations, and ending with the WSRF standard. On the web line the most important web service protocols are given, such as Hypertext Transfer Protocol (HTTP), the WSDL, and the set of Web Service (WS) related protocols (such as the Web Service Distributed Management (WSDM) protocol).

The WSRF has been described in five specifications. Conceptually, these specifications describe the same functionality as OGSI did, but with some changed syntax (e.g. to exploit WS-Addressing) and using different terminology. These WSRF specifications are described as follows:

**WS-ResourceProperties** defines how the type definition of a WS-Resource can be associated with the interface description of a web service, and message exchanges for retrieving, changing, and deleting WS-Resource properties.

**WS-ResourceLifetime** defines mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a resource, either immediately or by using a time-based scheduled resource termination mechanism.
**WS-ServiceGroup** defines an interface to heterogeneous by-reference collections of web services. Service groups can form a wide variety of collections of services, including building registries of services.

**WS-BaseFaults** defines a base fault XML types for returning faults in a web services message exchange.

**WS-RenewableReferences** defines a conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.

On top of these five WSRF specifications, **WS-Notification** defines mechanisms for monitoring WS-Resource changes. With **WS-Notification** it is possible to build a publish/subscribe notification mechanism.

### 2.7 Grid Middleware

A grid can be seen as an infrastructure of computing resources. On top of this infrastructure there is a need for middleware that copes with the differences between the computing resources and manages requests for computational power. Many different parties from the grid community have developed grid middleware solutions for their own needs. These middleware solutions are incompatible and each uses different protocols, components and interfaces. This results in slightly dissimilar functionality provided to the end-users. The Internet is one completely interoperable network, but computing grids are not. Simply put, computing grids can be separated in non-interoperable single-purpose islands.

Grid middleware is responsible for access to a grid resource, for communication to the user, for scheduling, for error handling, and for security. The middleware is the most important part of a grid, as it manages all communication and hides the differences between all connected entities.

Three of all the grid middleware environments are important for Dutch Space, since they are preferred by the company’s customers and have the highest applicability for the space industry. These middleware environments are candidates for integration into GridAssist, and therefore a short description is provided below.

#### 2.7.1 Globus Toolkit

One of the most popular grid middleware implementations is the open-source Globus toolkit [30]. It which has been continuously in development since 1998 by research institutes (e.g. National Center for Supercomputing Applications, Univa) in col-
Grid Middleware

2.7. Grid Middleware

laboration with some universities (e.g. University of Chicago and University of Edinburgh). The goal of his organization, called the Globus Alliance, is to develop fundamental technologies behind grids. After the first version of the toolkit, Globus Toolkit version 2 (GT2) was released in 2002. It contained no new functionality but was an improvement of the previous release. This version is so widely used that is can be seen the standard in grid middleware. The toolkit provides a set of modules for all grid related functionality.

The Grid Resource Allocation Manager (GRAM) provides an interface for accessing remote system resources. GRAM can be used for executing jobs, for scheduling, and for control. The underlying local systems of a grid may use different schedulers, processors, queuing systems, but grid clients only need to use the GRAM protocol.

For data retrieval the Globus toolkit provides a GridFTP protocol. GridFTP is based on the File Transfer Protocol (FTP) protocol but has some additional features which are indispensable for use in grid environment. These features include: third-party transfers, partial file transfers, Grid Security Infrastructure (GSI) and multiple data channels for parallel transfers. Another transfer protocol is Global Access to Secondary Storage (GASS), which is used to redirect standard output of a job over the network. This simplifies the running of applications that use file transfers. With GASS it is for example possible to redirect the standard output of an application to another grid node, or to a client application.

A third example of a Globus module is the Monitoring and Discovery Service (MDS). This service provides a mechanism for publishing and discovering resource status and resource configuration. With MDS it is possible to automatically discover a resource on a grid matching your needs. MDS is based on the Lightweight Directory Access Protocol (LDAP).

As mentioned earlier, security is a very important aspect when providing resources to the outside world. The Globus Toolkit uses GSI, which provides secure communication between any two parties in a grid, with some additional features such as single sing-on and mutual authentication. GSI is based upon X509 certificates, public key infrastructure, and Secure Socket Layer (SSL). With each request to the Globus grid node, the user sends along a proxy certificate. A proxy certificate is a copy of the original X509 certificate that is valid during a short time period. The grid node that has a user’s proxy certificate can act on the user’s behalf, i.e. be authenticated and authorized as if being the user. This security mechanism allows grid nodes to submit jobs to other grid nodes, without any additional communication with the user (i.e. dynamic discovery of resources and for dynamic job allocation).
When the Globus Alliance released their second version of the toolkit, they were in fact pioneering grid technology. For the first two versions of the Globus toolkit no one was concerned with standardization. But at the time of the third version, the GGF had proposed the OGSA, which was specified in OGSI. In the third version of the Globus toolkit, all services were ported to grid services. This means that the services that the Globus toolkit provides are accessibly through a web service interface. Also some web service related components were added to the third version of the toolkit, such as the Open Grid Service Architecture-Data Access and Integration (OGSA-DAI). This new service can be used to access data that is separated over multiple data sources on a grid.

Unfortunately, OGSI was not going to be the standard that GGF had hoped for. The GGF worked on an improvement called WSRF, the Globus Alliance followed with the fourth version of the toolkit. This version includes web service components that can be divided into five categories: security, data management, execution management, information services, and common runtime components. Globus Toolkit version 4 (GT4) is not a completely redesigned toolkit, but rather a refactoring of the previous release to WSRF.

The major difference between the second and the later versions of the Globus toolkit lays in the fact that all functionality of the newer toolkits is basically implemented as services (a grid service in Globus Toolkit version 3 (GT3) and a web service in GT4). This is called a Service Oriented Architecture (SOA) [46]. One of the advantages of this setup is that it is easier to implement a new service and all services use standard protocols.

2.7.2 gLite

The Enabling Grids for European E-Science (EGEE) project refactors several European grid solutions into one grid middleware implementation for European scientists [16]. This middleware is called gLite, and is based on experiences gained with existing (and previous) middleware of various other grid projects. The gLite middleware architecture is largely based on Large Hadron Collider Computing Grid (LCG) and the European Data Grid (EDG). gLite is supposed to be lightweight and should be easy and quickly to deploy, but in my experience this toolkit is (still) not easy to deploy and requires a lot of resources.

gLite uses a Service Oriented Architecture (SOA) based on standards developed within the web services community. SOA refers to a method for combining interacting services into a reliable distributed system. A fundamental aspect of SOA is
modifiability; one service might change its inner workings, but should still be able to communicate through the same interface. GLite is currently not fully implemented according to the WSRF specifications. But the gLite architecture is using web services and has been designed with future migrations to WSRF in mind.

The services of the gLite middleware are depicted in figure 2.2, each of the services should be deployed in a separate machine, but it is possible to combine multiple service on one machine. The Workload Management Service (WMS) can communicate with multiple Computing Element (CE)s, and each CE can communicate with multiple Worker Node (WN). Which gives this setup some redundancy and allows CE and WN provide specialized functionality over its counterparts.

The WMS and the CE are the most important services for GridAssist as they provide basic functionality for job submission. The WMS functions as a scheduler of tasks over multiple grid resources. Its goal is to schedule the tasks as conveniently, efficiently and effectively as possible. The CE service represents a computing resource, the main functionality that is exposed is job submission and job control [16]. These two components should provide a grid with the minimal functionality that GridAssist uses: job submission, job state monitoring, and file transfer. The other components of the gLite grid middleware:
**IO-Server** allows posix-like access to files stored in the Storage Resource Manager (SRM). It supports GSI authentication, authorization and name resolution.

**Local Transfer Service (LTS)** provides a file transfer and file placement service and is also used for moving files.

**Catalog** is a database that stores meta-data; it keeps track of the mapping between virtual file names and the logical file names. The catalogs are also accessible for the user; it allows management of schemas, setting and getting of values of attributes and execution of queries, and management of the access permissions on each item.

**Relational Grid Monitoring Architecture (R-GMA)** server accepts connections from clients, published for example by services user jobs, and forwards the information to the appropriate consumers. This service has been built on the publish-and-subscribe paradigm. Because R-GMA can publish information from all kinds of sources, it needs input from all other services.

**Logging and Bookkeeping server** keeps track of the job status information. This service is used by all other services that deal with the state of a job. The retrieval of job states and raw events are handled by the WS querying interfaces.

**Worker Node (WN)** hosts all the necessary clients to interact with the grid middleware from within a job. Currently, clients for the IO Server, Logging and bookkeeping, Relational Grid Monitoring Architecture (R-GMA) and WMS are provided.

### 2.7.3 Uniform Interface to Computing Resources (UNICORE)

Uniform Interface to Computing Resources (UNICORE) was started in 1997 by German universities and computing centers [23]. This grid middleware has no connection with the Globus toolkit, it has been created from scratch. UNICORE tries to hide the seams between computing nodes caused by; different hardware, incompatible system software, and historically grown computer center practices. To do this, the UNICORE grid is build upon three tiers (see also figure 2.3):

- The top tier is formed by the client application that interacts with the user. The standard client delivered by UNICORE supports the definition of workflows in direct acyclic graphs. This part should eventually be incorporated into GridAssist. Communication with the tier below is done with the UNICORE Protocol Layer (UPL) that is built on top of SSL and defines how data is sent to the UNICORE gateway.
2.7. Grid Middleware

- The middle tier is the UNICORE Site (USite) that accepts incoming execution and/or data requests from clients. The USite has a Gateway that acts as a communication port for the clients. The gateway is responsible for the authentication of users and grid nodes. A USite consists of one or more Virtual Site (VSite) of the lowest tier.

- The lowest tier is concerned with the actual grid nodes. Each VSite represents an actual execution and/or storage system. A VSite has a Network Job Supervisor (NJS) to which the Gateway sends requests. This NJS authorizes the user and starts one or more Target System Interfaces (TSI). The TSI is the working horse of the UNICORE grid, and is the interface to the batch subsystem.

UNICORE uses X509 certificates for authentication, but makes no use of proxy certificates. The certificate implementation is much more static, which leaves no room for dynamic discovery of resources and neither for dynamic job allocation. On the other side, this security implementation is safer since no one else can acts on the user’s behalf.

The UNICORE client consists of two components, the Job Preparation Agent (JPA)
and the Job Monitoring Controller (JMC). The JPA is responsible for sending Abstract Job Objects (AJO) to the gateway. AJO is a JAVA class library, which lays the basis for specifying jobs and client-server communication. AJO can be divided into three components; abstract task object hierarchy for the description of jobs that are to run on a grid, the resource object hierarchy for the description of resources, and the UPL for communication between the client and the gateway [23]. The JMC submits Abstract Status Request (ASR) to the gateway and displays job states to the user. ASR’s are in fact also part of the AJO class library.

2.7.4 Grid Middleware Comparison

In order to make the comparison of the four middleware environments somewhat easier I have provided a summary (see table 2.1) in which the important aspects of each grid have been described. A lot of different protocols have been developed which are practically alike. For example, the job representation languages can easily be translated into one another. Only the format differs, GT4 uses an XML based language, UNICORE uses java objects to describe a job and the others also have their own representation.

The middleware environments are particularly alike in their security mechanism; they all make use of X509 certificates to authenticate users. One exception is that gLite uses a Virtual Organization Management System (VOMS) extension in its certificate to indicate the VO that the users belongs to. And furthermore, UNICORE does not use proxy certificates. But still, the main security difference lies in the authorization mechanism. The Globus middleware environments both use a gridmap file that maps X509 identities to a local user account. GLite does this mapping inside the VOMS server, but also makes a mapping of a VO to a local user account. UNICORE uses a database that holds the mapping of a certificate identity to a local account, instead of a file.

When more computing power is necessary, its logical that more computing elements are required. But some grids require multiple machines just for basic operation. In contrast to the Globus toolkits (which can use one grid node for all services), UNICORE and gLite need multiple grid nodes for their services. UNICORE needs a Gateway one (or more) NJS and multiple TSI’s that interface to the hardware. The gLite grid needs about six or seven machines for the most distributed deployment, but some services can be deployed together.

A dynamic aspect of grid middleware is the discovery of grid nodes at run time. This is currently not used by GridAssist, but it can save the user from the effort of adding, removing and editing resources in GridAssist’s registry. Dynamic discovery
### Table 2.1 — Grid middleware comparison

<table>
<thead>
<tr>
<th>Aspect</th>
<th>GT2</th>
<th>GT4</th>
<th>gLite</th>
<th>UNICORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardization</td>
<td>None</td>
<td>WSRF</td>
<td>Nearly WSRF</td>
<td>OGSA compatible</td>
</tr>
<tr>
<td>Job representation</td>
<td>RSL</td>
<td>XML</td>
<td>ClassAd (JDL)</td>
<td>Java Objects</td>
</tr>
<tr>
<td>Authentication</td>
<td>X509 cert proxy</td>
<td>X509 cert proxy</td>
<td>VOMS X509 cert proxy</td>
<td>X509 certs no proxy</td>
</tr>
<tr>
<td>Authorization</td>
<td>Gridmap</td>
<td>Gridmap</td>
<td>VOMS</td>
<td>UUDB</td>
</tr>
<tr>
<td>Data management</td>
<td>GridFTP</td>
<td>GridFTP</td>
<td>GridFTP subset</td>
<td>Java Objects</td>
</tr>
<tr>
<td>Deployment</td>
<td>1 node</td>
<td>1 node</td>
<td>multiple nodes</td>
<td>multiple nodes</td>
</tr>
<tr>
<td>Dynamic Discovery</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

is, due to its strict security policy, not supported by the UNICORE grid middleware.
The grid is based on very complicated technology, users might not have the experience and time to familiarize themselves with its protocols, terminology and inner workings. But a grid has real potential for users who demand high computing power and provides them a chance on business advantages. As mentioned in the introduction, GridAssist can be seen as the user’s viewing window of a grid. It has been developed as a user-friendly application that allows scientists to use grid functionality without the need to understand the details of the underlying machines and grid middleware environments.

3.1 The Flow of Work

GridAssist is mainly used by scientists to define units of work that can be submitted to a grid. These units of work are jobs, which are defined as workflows: they describe the flow of data between multiple jobs and storage locations. A job in a workflow can either be the execution of an application or the transfer of files to a storage service. Each single execution of an application or the transfer of files on a grid is called a job process and each node in the workflow is called a job.

A link between two nodes in the workflow is coupled with the transfer of one or more output files. Due to the fact that an executable can have multiple output files, it is possible that a job node consists of multiple job processes. For example, if we have a simple workflow of service A and B, which are connected by a link from A to B. If service B is a storage service, the output files of service A are transferred to a grid node of service B and stored. However, if service B is an application, the files
produced by service A are copied to a temporarily folder and service B is executed on each of these files. This mechanism simplifies the coupling of applications that produce multiple output files.

An example of a workflow can be seen in figure 3.1, which is a screenshot of the workflow tool. It shows how a workflow for processing OMI data could look like. This specific screenshot shows the workflow pane, where users can define workflows and submit them to a grid. The workflow in this screenshot consists of six services (applications), one storage node for input data and one for storing the output of the workflow. It can be seen that the data is first filtered, then corrected for errors and then processed. The pre-processing service splits the data, which allows the processing to continue in two levels. The post-processing combines the data and generates the final output, that can be in the form of a picture (see figure 1.1).
3.2 Design for Assistance

GridAssist has been designed according to the thin client and fat server paradigm, see figure 3.2. The client component, called the workflowtool, consists of a graphical user interface that allows the user to easily define workflows, perform data transfers and send other commands to a grid. Furthermore, the workflowtool visualizes the status of submitted jobs and graphically displays the resource usage. The workflowtool provides an understandable view on jobs and resources of a grid. Its functionality has been divided into three parts. This can be seen in figure 3.1 in the three tabs in the left top corner.

- The registry panel allows users to administrate resources, applications, storage directories, users and middleware types. This panel is especially useful for GridAssist’s administrator.

- The workflow panel allows users to construct workflows from the applications and storage directories as entered in the registry. Workflows can be created, edited and submitted to all or some specific grid resource.

- The monitor panel allows the user to monitors the submitted jobs. Progress reports on the job processes and the workflow completion can be viewed here.

In the design of GridAssist, there has been a particular focus on the usability aspect of the workflowtool. It is important that complex jobs can be modeled and submitted in an intuitive way. The focus of usability was specifically on hiding all complex protocols and service of the computing grid from the user. Furthermore, the workflowtools provides users with a mechanism to easily create complex workflows.

All grid requests are translated into XML and are sent to the server component, called Controller. The Controller is the workhorse of GridAssist, it services multiple users (i.e. Workflowtools) and performs their requests by communicating to a grid. There are two sub-components that work together: the ControllerInterface and the WorkflowEngine. The ControllerInterface sub-component receives, translates, and handles requests from the Workflowtools. Newly submitted jobs are stored in a database, and the job’s status is retrieved from the database. The WorkflowEngine fetches new jobs from the database and stores changed job statuses in it. Furthermore, the WorkflowWngine is the workhorse of the Controller component; it submits jobs, fetches output, monitors the status of running jobs and executes file transfers.

Both the ControllerInterface and the WorkflowEngine use a wrapper (which is designed for Globus toolkit version 2) for communication to a grid. Originally, this component was meant to extent the functionality of the Commodity of Grid
toolkit (CoG) kit\textsuperscript{1}. This kit is a java API that allows communication with grid nodes. Unfortunately, the CoG kit is currently only available for Globus toolkit version 2, so communication with other grid middleware cannot be established through the CoG kit. This Wrapper performs calls, through the CoG kit, to the grid node in order to submit jobs and transfer files.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{gridassist_components.png}
\caption{GridAssist’s components}
\end{figure}

The workflow engine of GridAssist submits a job process to a grid when either it has no input files, or when all the predecessor job processes from the workflow have finished executing and the necessary files have been transferred to the grid node. The workflow engine continuously checks the waiting job processes list for items that have met their prerequisites and are ready for execution. The workflow engine spawns a new thread for each job process submission to a grid. This threaded structure greatly increases the performance of the component. When a job process has been successfully submitted to a grid, the workflow engine keeps polling the state until it retrieves either a finished or a fault state. This state information is stored in the database, and is later retrieved by the controller interface for visualization at the client side. A job’s life in GridAssist goes through the following states: waiting, running and ends as either failed or finished.

\textsuperscript{1}See http://www.cogkit.org
3.3 Communication

The interface between the controller and grid nodes only supports communication with GT2 middleware. The controller uses the GRAM service (discussed on page 17) to submit jobs that are specified in the Resource Specification Language (RSL). For the redirection of standard output of executables the GASS service is used. And for file transferring between either the grid node and the controller, or between two other grid nodes (third-party transfer), the GridFTP protocol is used. GridFTP has been developed by the GGF as the standard file transfer protocol for a grid. It is based on the FTP protocol, with some extensions (see section 2.7.1).

Communication between the workflow tool and the controller is done by the Apache Axis SOAP engine\(^2\). SOAP stands for Simple Object Access Protocol, and is used to translate client side method calls into method calls to the controller class. This allows us to call methods of an object that is located on a remote machine. Axis sends back the results of the called method in a Simple Object Access Protocol (SOAP) (XML) message to the workflow tool.

3.4 Further Evolution of GridAssist

This section describes aspects of GridAssist that might still need development. Furthermore, a discussion of future developments is given.

A grid’s goal is to hide the fact that it consists of many different nodes and to provide services as if provided by a single big grid node. Currently, GridAssist uses grid nodes which are scattered and un-interoperable, in combination with small cluster grids. Therefore, the scheduler of GridAssist communicates with multiple independent resources and performs scheduling on its own. This project is to make the scheduling among heterogeneous resources a possibility. But when the grid technologies evolves and form a more interoperable entity, the scheduling among resources should be done by a grid. A grid can schedule in more dynamic way by taking CPU load and the amount of free memory into account. This can optimize the performance of the workflow execution. In my opinion, the scheduling among available resources is (currently) a strong point of GridAssist, which makes it possible to handle different un-interoperable grid nodes. But this ’self-scheduling’ but will in the future loose its value.

Furthermore, an improvement might be possible on the design and philosophy of GridAssist. In the current situation new resources should be added manually before

\(^2\)Axis is a framework for constructing SOAP processors such as clients, servers and gateways. See http://ws.apache.org/axis
the workflow engine schedules job processes to it. Current grid middleware supports
the dynamic discovery of new resources, GridAssist just does not use this function-
ality. A discovery service, which is actually provided by grid middleware such as
Globus, can be used to let GridAssist automatically discover new resources.
Chapter 4

Designing Interoperability for GridAssist

This chapter will describe the process of designing the adaptations to GridAssist. Design documentation is done here in a brief way, for a more complete overview of the design of the adaptations I refer to the design documents [10, 11, 12, 13].

The design was performed according the software design standard for space community as proposed by European Space Agency (ESA). This protocol, Procedures Standards and Specifications for Software Engineering (PSS-05) [2], describes the whole software engineering process from requirements elicitation to testing and writing a user manual. The complete PSS-05 standard consists of eleven parts describing standards for all ESA’s engineering related activities. The PSS-05 standard has been recently replaced by a general European standard, called the European Cooperation for Space Standardization (ECSS). However, the ECSS is very general and should be tailored each time it is used. The PSS-05 standard, on the other hand, contains a version for small projects, which makes it more suitable for this project. And furthermore, there is more experience within Dutch Space with the PSS-05 standard, which also favors its choice. In addition to the steps from PSS-05, the assessment step has been added for testing the software architecture to the requirements (see figure 4.1).

The design process started with a requirements elicitation phase that analyzed the needs of the stakeholders of GridAssist. Requirements from previous versions of GridAssist were analyzed for consistency with the interoperability requirements. This phase is, according to the PSS-05 standard, divided into user requirements
specification and software requirements specification. The development of the interoperability design and the adaptation of the original design were divided into two phases; architectural and detailed design. In the architectural design phase the components of the software were identified and also the interactions between them. This phase provided a preliminary design on the adaptations to GridAssist. The software architecture design delivered by this activity provides valid input for the scenario-based software architecture assessment methods. The assessment methods are to verify, before detailed design, if the quality attributes as stated in the requirements document are supported in a well enough manner (see chapter 5 and 6). With a positive outcome of the architecture assessments, the architectural design of GridAssist was worked out more precisely to give a detailed description for the implementation phase. Detailed design is the input for writing a test plan and for implementation.

4.1 Requirements Specification

The requirements of the design phase can be seen from two perspectives: from the user’s and from the system’s point of view. The user is concerned with what functionality is provided by the system. From the system’s point of view, the focus is more on how to provide this functionality. The user requirements (i.e. what the user expects the system to do) influence and lay the basis for the system requirements (i.e. how the systems is expected to do what the user expects). Appendix B gives an overview of the requirements of the adaptations to GridAssist. For a complete overview of all requirements consult the design documents [13, 12]. These requirements were obtained from the designer of the original GridAssist, who is also
4.1. Requirements Specification

an experienced user.

4.1.1 User Requirements

These requirements (see table B.1) describe the functionality that the user expects from the adaptations to GridAssist. Functional user requirements are derived from the use cases, and simply originate from the question: What does the user want to do with GridAssist? The subjects of the user requirements are basically the main functions of GridAssist; workflow execution, resource management and file transferring.

The quality user requirements (see table B.2) describe the quality of the functionality that the user expects from the adaptations to GridAssist. In this case, the requirements are mostly concerned with preventing the degradation of the current provided quality. It can be seen from the user requirements that usability, portability, modifiability and performance are important. In order to assure the fulfillment of these requirements, a scenario-based assessment will be performed after architectural design.

4.1.2 Software Requirements

Functional software requirements are the product of the translation of user requirements into requirements on the application that is going to be adapted (see table B.3). The user wants to be able to use multiple middleware environments in job submission, file transferring and job monitoring, which results in software requirements describing the system in light of a framework for wrappers to each of the middleware environments (see SR01 and SR02). As was already mentioned in chapter 2, there are three candidate middleware environments: GT4, UNICORE and EGEE’s gLite. These three, plus the currently supported GT2 middleware environment, need a wrapper which handles the communication, job submission, and job monitoring (see requirements SR03, SR04, SR05 and SR06). These wrappers for middleware environments should be plugged-in to the controller of GridAssist.

Analogously to the user constraint requirements, the quality software requirements describe the quality that is expected from the system’s point of view (see table B.4). From these requirements, a decision was made on the subject of the software architecture assessment. Performance, security, usability, portability and modifiability are attributes that are present in the requirements. Because GridAssist’s strength is based on the simplicity of the GUI, usability is a really important quality attribute.

These requirements were analyzed for identification of several use cases and sce-
narios, which are used in the design and assessment process. The use case 'change resource' is given as an example, a complete set of use cases and scenarios can be found in [13]. The 'change resource' use case can be described as follows: change a resource that is listed in the registry, thereby changing name, addresses, type of job manager, length of queue, number of simultaneous running jobs and middleware type (e.g. GT4, UNICORE). This use case is an administration task and should only be executed by a user that has administrator rights. An example scenario that can be used to extract the system’s design properties and assess quality attributes is given below. The participating actor is an administrator called Bob.

1. Bob has got information that the new grid cluster is not running Globus Toolkit 4, but UNICORE. This was a wrong statement in the definition of the cluster. Due to this mistake, a wrong protocol is used for communication with the cluster. No jobs can currently be run on the cluster and Bob wants to solve this problem.

2. Bob starts up the GridAssist workflow tool creates a valid credential and goes to the registry panel. There he clicks on the resource he wants to change. Information regarding this resource is shown: its name, address, middleware type and number of jobs it can run.

3. Bob changes the middleware type field from 'Globus4' to 'UNICORE'.

4. After that, Bob clicks on apply and saves the registry. Now, the GridAssist controller will use the right protocol for communicating with the grid cluster.

4.2 Architectural Design Solutions

The architectural design phase translates the software requirements into a rough software design. This architectural design is based on components and the relations between them. Furthermore, in order to analyze the information flow and relations between the components, the component interactions are identified. In a later phase this architectural design is translated into a more detailed design.

In order to satisfy the requirements as mentioned in the previous sections, redesign of GridAssist is needed. Three solutions are discussed below. From these solutions the best solution is chosen (see section 4.2.4). Each of the design alternatives has a specific component for the grid middleware environments. The difference between the solutions lies in how these middleware specific components are integrated in the original architecture of GridAssist.
4.2. Architectural Design Solutions

4.2.1 Multiple Controllers
The first solution consists of adapting a whole controller component for every specific grid middleware environment. This requires a duplication of the controller for each middleware environment, as can be seen in figure 4.2. The advantage of this solution is that the original controller of GridAssist does not need any adaptations and remains operational. Furthermore, the design of the additional controllers does not concern the legacy design of the controller and is therefore not restricted in any way. This means that the controller can be specifically designed for the middleware environment. This solution also has some drawbacks, because it would mean that the workflow tool should hold connections to multiple controllers, which is currently not the case. When the user wants to communicate with grid nodes that use another grid middleware environment they would need to connect to another controller component. Another drawback is that this setup does not support the execution of workflows that run on multiple different middleware environments (i.e. inter-middleware workflows). Furthermore, because each controller needs a database, working directory and CPU cycles, there is a higher resource requirement.

![Figure 4.2 — Design solution: multiple controllers](image)

4.2.2 Middleware Wrappers
The second design solution does not require a controller component for each middleware environment. It only needs one generic controller that uses multiple wrappers for middleware communication. A wrapper is specific for one grid middleware environment and handles all communication to and from the grid nodes. The usage of wrappers is dynamic; the user can make on-the-fly associations between wrapper classes and middleware types. In this way it is also possible to have multiple different wrappers for one specific middleware environment. Only one wrapper for
a middleware environment can be used at a time, but the user can switch between them as he sees fit. As long as the wrappers’ compiled class and the libraries are copied to the controller component. A loader class has been added to load these plug-in wrappers and provide them to the other components. Figure 4.3 depicts the adapted component structure of GridAssist, where the internal relation between the components can be seen. Each grid node that has a different color resembles a grid node that runs a different grid middleware environment.

This setup (see figure 4.3) makes the usage of multiple grid environments very flexible and can be completely hidden from the user. Because all information concerning grid middleware and workflow execution is located in one location, inter-middleware workflows are supported.

![Diagram](image)

**Figure 4.3 — Design solution: middleware wrappers**

However, this design solution requires some redesign of the original controller component, which always lays restriction on the possibilities of the middleware wrappers. Furthermore, all wrappers need to implement the same functionality through a generic interface. This means that GridAssist can only use the functionality that is in the conjunction of all functions provided by all grid middleware environments.

### 4.2.3 A Hybrid Solution

This solution is in fact a combination of the two above. It uses multiple controller components that in turn each use multiple wrappers. With this solution it is possible to enjoy the best of both worlds. On one side, there can be a controller that is specific for one type of grid environments, and on the other hand, each controller can use a wrapper for grid specific communications. An example of this setup would be to
create a WSRF specific controller that has a gLiteWrapper and a Globus4Wrapper, and a legacy globus controller that uses wrappers for Globus2 and Globus3. In this way, functionality can be separated in two steps: in controllers and in wrappers.

Although this setup will allow inter-middleware workflows for all wrappers in one controller, true interoperability is not guaranteed as long as there are multiple controllers. Furthermore, still many resources are required and the user is still bothered with controller switching when communication with another grid middleware environment is required.

![Design solution: hybrid](image)

**Figure 4.4 — Design solution: hybrid**

### 4.2.4 Discussion of Design Solutions

For reasons of true grid middleware interoperability, less resource requirements and true support for inter-middleware workflows, the second design solution was preferred above the other two. Furthermore, from a usability perspective, it is better to hide the fact that there are multiple controllers from the user. The user just wants to run his job, and does not care on which middleware environment this is realized.

The selection between the three design solutions would have been suitable for a software architecture trade-off assessment (described in chapter 5), but due to the one controller restriction the choice was already clear and a time consuming method (such as Architectural Trade-Off Analysis Method) would be an overkill for a relative simple choice. On the other hand, a full scale trade-off assessment allows the identification of trade-off points and can indicate unexpected advantages of an un-
derestimated design alternative. But the large requirements for a trade-off method (participation of people and time) undermined its use in this case. Architecture assessment methods are introduced in chapter 5 and used for verification of quality attributes in chapter 6.

4.3 Architectural Design Details

In this section the functionality of the components of the ’middleware wrappers’ design solution is described in more detail.

4.3.1 WrapperLoader

This component retrieves the wrapper class name from the database and loads the wrapper class from a predefined location. The WrapperLoader returns a wrapper object that is used by the ControllerInterface and the WorkflowEngine for communication with grid nodes. The WrapperLoader has only one function: `loadWrapper(int middlewareType)`, which returns an instance of the wrapper class that is used for the middleware type.

4.3.2 Wrapper

For each grid middleware environment that GridAssist communicates with, a Wrapper object is used to translate function calls into calls to the services of the grid middleware on the grid node. This name was adopted from the original GridAssist, Grid Interface should be better describing this component’s function. If for example a job is to be submitted to a grid node that has Globus Toolkit 4 installed, the WorkflowEngine retrieves a Globus4Wrapper object from the WrapperLoader and calls the submit function of this Wrapper object. The Wrapper connects to the specified grid node, makes sure a valid certificate is present, translates the job into the appropriate protocol and submits the job.

In order to achieve true interoperability each wrapper needs to implement the same functionality. This was realized by creating an interface class that defined the necessary functions for all wrappers. Each wrapper class implements this Wrapper interface class. The following functions provide all the functionality needed by the WorkflowEngine and the ControllerInterface. The wrapper interface class defines exactly these functions.

    boolean checkCredential() Checks whether the credential is present and that it is still valid, if so it returns true, else false.
void copy(String src, String dst) Copies a file from source (src) location to the destination (dst) location. This function can be used to upload input or download output. When the source and destination urls point to two external locations the transfer is called a third-party transfer. This function currently only supports third-party transfers that use one protocol. But these inter-middleware transfers can be divided into two two-party transfers. One from source to temporarily local folder, and one from this local folder to the destination.

void copy(String srcHost, String srcFile, String dstHost, String dstFile) This is just a convenience function, it rallies calls to the function above.

String getName() For identification purpose, this function returns the middleware name where this wrapper is designed for.

String retrieve(String url) Output of the execution of programs is always placed into files. This function retrieves this output by transferring copying the files and returning its contents as a string.

void run(String url, String jobManager, String executable, String[] args, String output, String dir, String jobType, String queue) Runs the job specified by the executable and args on the resource specified by url. The local handler for the job is defined by jobManager (e.g. condor, pbs, fork, lsf). The output of the executable is stored in file with name output.

void run(String url, String rsl) Runs the job specified by rsl on the resource specified by the url. The rsl should specify the path of the executable, the arguments, where to place the output files, and the execution directory.

void setCredential(Credential c) If c is valid, this function sets the credential of this wrapper to c. This Credential is a superclass of either GlobusProxy, MyProxy, or a X509Certificate credential.

void setCredential(GSSCredential c) If c is valid, this function sets the credential of this wrapper to c.

void setCredential(InputStream[] f, String passphrase) This function generates a credential from the files in f and the passphrase. It sets the credential of this wrapper
void stopRunning() If this wrapper is currently running a job on a grid node, it is stopped.

4.3.3 Component Interaction

The user uses the WorkflowTool to submit requests the ControllerInterface. For requests like file transferring, retrieving a file list and determining the working directory the ControllerInterface communicates directly to a Wrapper. For requests that require some scheduling (job submission), the ControllerInterface inserts the job into the database. The WorkflowEngine runs continuously fetching any new entries from the database, scheduling the job to a resource, and submitting the job.

Whenever the ControllerInterface or the WorkflowEngine need to communicate with a grid resource, they retrieve the resource’s middleware type from the database and send a request to the WrapperLoader to get a Wrapper object for the resource. The WrapperLoader checks the database for the Wrapper class associated with the middleware type, loads the class, instantiates it and returns the object. Whenever the WorkflowEngine or the ControllerInterface receives the Wrapper object, they set the Wrapper’s credential to that of the user for whom the request is performed. After that, the wrapper is authenticated and authorized to access the grid resource, in the same way the user is. In fact, the Wrapper acts on the user’s behalf since it has all his credentials. After the security has been set, the Wrapper can be used to submit jobs, retrieve output, transfer files, and to monitor the status of running jobs. When the WorkflowEngine has retrieved some new information (e.g. a changed job state or a partial completion of a workflow) from a grid, it inserts this information into the database.

4.4 Further Design Steps

The architectural design was followed by detailed design, which is to further specify the design into class diagrams, sequence diagram and components. This results in a description of the system that is detailed enough for the implementation phase. Detailed design is the basis for writing a test plan and for implementation. Further information about this design phase can be found in the detailed design document [11].

After detailed design, but before the implementation phase, the testing specifications and testing goals are defined in a test plan. This plan defines how to verify the implementation according to the requirements. After that, based on the detailed design, the source code of GridAssist is adapted. See for example the Wrapper
4.5 Testing

After implementation of the adaptations to GridAssist, some tests were performed to check the fulfillment of the requirements. This was done as depicted in figure 4.5. Unit testing verifies the design and implementation of components from the lowest level defined in the detailed design up to the lowest level in the architectural design. Unit tests check whether the module is doing what it is supposed to do (black-box testing) and that it is doing it in the way it was intended (white-box testing) [52]. After the unit testing phase, the integration between the components (input-output relation) is tested. The system testing phase includes operating the system and checking if all functionality has been implemented as was required. The testbed testing phase was performed on a small grid (4 resources) and included a performance test, an inter-middleware workflow test, an inter-middleware transfer test, and monitoring test of the controller’s memory usage.

All the above mentioned testing activities have as goal to verify that the design does match the requirements. This was the case for the majority of the requirements. However, unit testing has discovered that a performance related requirement was not satisfied in the design. After tracing back the design decision, it became clear that this requirement was omitted for security reasons. Another test result, showed that the GUI elements have increased from 90 to 112 elements (see also table 6.5), this is a relative increase of 24% (20% was required as maximum). Integration testing finished without any problems, and system testing did not encounter any serious problems.

For complete testing in a natural environment, GridAssist has been deployed on a test system. In this environment, GridAssist performed certain activities while its behavior was carefully observed. First, a test workflow that required multiple resources of different middleware types was submitted. Thereafter a transfer test that includes two transfers is performed, one from a GT4 grid node to a GT2 grid node and the other vice versa. These two test runs did complete without any issues, and showed that the required functionality for workflow submission was implemented correctly. The third testbed test focused on the performance difference between GT2 and GT4. For a valid comparison of the performance of the two toolkits, a sample job was submitted to the toolkits running on the same physical machine one

---

1 A tool to generate source code documentation through comments in the source code.
at a time. The sample application is a simple 'sleep' job that runs (i.e. sleeps) for 5 seconds. Roughly all time above this is overhead of GridAssist and the Globus Toolkit. So because hardware, GridAssist version and the job are identical, the only variable is the middleware environment and the wrapper for it. The result of this performance test showed that job completion is 3 seconds slower on GT4 than on GT2, for a 60 second running job (see figure 4.6). Normal jobs will have a much longer running period, so the overhead will be neglectable. Remarkable is the longer running time for the first job. This is caused by the Apache Axis engine, who needs to create objects when it is first called. This requires some additional time, which is equal for GT2 and GT4. Once the objects are created, they stay in memory and will be used for subsequent calls.

Testing was concluded with a monitoring session of the Java Virtual Machine (JVM) with JProfiler\(^2\). This is a monitoring tool that allows fine-tuned profiling of memory, CPU, allocation, threads, etc. I have used JProfiler for memory leak detection, and examining the heap size increments as jobs were submitted. Unfortunately, there is a small leak that was also detected in the original version of GridAssist, but which is probably caused by invalid garbage collection, which is handled by the JVM.

### 4.6 The good, the bad and the ugly

This section gives some thoughts on the design; what is good and what could be improved. Further reflection on the support of usability, modifiability and portability

\(^2\)See [http://www.ej-technologies.com](http://www.ej-technologies.com)
4.6. The good, the bad and the ugly

Figure 4.6 — Grid Middleware Performance Comparison

is given in chapter 6.

4.6.1 Reflection on Design

As a prototype of the design, a wrapper for the GT4 was produced (see appendix C for its source code). Originally, GridAssist supported only GT2. The choice for GT4 was made, because it has the highest applicability for GridAssist. Third party transfers (transfers between two grid nodes that have different middleware) are easier in this situation, because GT2 and GT4 both use the same transfer protocol (gridFTP). The other two grid middleware candidates (UNICORE and gLite) both also support gridFTP. This protocol is a standardization initiative of the GGF, but not all grid middleware environments currently use this standard. It is thinkable that future wrappers of GridAssist use other transfer protocols and third party transfers will require the use of two different protocols. This would mean that the transfer should be performed in two steps; first transferring the files from the source grid node to the controller, and the second step transferring the files from the controller to the destination grid node. This will significantly degrade the file transfer speed, and with that the performance of the workflow execution. Therefore, some care should be taken when trying to integrate a different transfer protocol.

4.6.2 A Comparison of Grid Middleware

Grids have come a long way in the last three years, the GGF has produced quite a lot standards and the web and grid developments have merged into each other
(see figure 2.1). But is also important to investigate the developments of the grid middleware environments; what improvements has the Globus Alliance achieved with its latest toolkit over previous versions? The list below describes the main differences between GT2 and GT4.

- The installation process, in respect to required time and expert knowledge, does not differ much between the two versions.

- Interfacing to GT2 can be done with a CoG kit. The CoG kit for GT4 was not delivered in time for me to use, so interfacing has been done with certain libraries copied from the GT4 installation. But with the CoG kit interfacing to GT4 should not be more complicated.

- It is currently very difficult to find grid solutions that are using GT4. This seems logical, since the final toolkit has been released for only three months. What can be concluded from the extremely busy discussion groups concerning the installation of GT4, is that many parties are indeed preparing for future adoption of the toolkit. I believe that GT4 is the grid middleware standard of the future. It is eventually going to replace the currently widely used GT2.

- Will a grid service that functions as a web service provide a lower performance? In order to answer this question, performance has been reviewed in the testing phase. This test showed that there is only a small difference of the execution time between GT2 and GT4. GT4 is only 3 seconds slower in completing a simple job, which will be insignificant for long running workflows. This difference can be either be caused by the grid middleware itself, or by the Globus4Wrapper that communicates with the middleware. It is possible, that usage of the CoG kit for GT4 will make the performance difference negligible.

Secondly, gLite was investigated for possible integration with GridAssist. This grid is focused on creating larger grids and is therefore more complex then the Globus Toolkit. This makes gLite harder to install and requires much more physical machines for providing basic functionality (job submission and data transfer). Unfortunately, the installation scripts are either unfinished or the product of bad design, because I noticed far more troubles when installing the WMS and the CE components then with the GT4. The following list gives some examples of incompatibilities that currently prevent easy adoption into GridAssist.

- gLite uses multiple C++ compiled libraries, which have been compiled for a specific Linux version that uses gLib version 2.3 and above. This lays restrictions on the systems that can be used for the Controller component of GridAssist. The Globus grids use Java, which makes them more interoperability in respect to platform choice.
• There is little to no documentation available concerning the Java Application Programmers Interface (API) of the WMS component. The VOMS server Java API is not even available at all. Communication with the VOMS server, in order to obtain a VOMS-enabled proxy certificate, should be done through native functions from Java to the C++ API. This makes the interfacing very complex and will possibly also decrease the performance of the gLiteWrapper.

• And furthermore, the gLite grid requires a specific deployment of services among machines. In my testbed I installed all services VOMS, CE and WMS on one machine, which led to incompatibilities and prevented an easy deployment of the services.

When gLite is fully operational it provides some additional features over GT4, such as bookkeeping of used computing power and storage space. The completeness and the distributed deployment makes this grid middleware environment extremely useful in the more demanding situations, such as the processing and distribution of data that is produced by the LHC. In general, I think the Globus grid is more useful for smaller grids due to the easy installation and the limited requirements on resources. The gLite grid, on the other hand, is more useful for creating larger and more complex grids and will require more installation effort and also requires more resources. I certainly disagree with the claim that gLite is a lightweight grid middleware solution.

In the ideal situation, the standards of the GGF should be used in each grid middleware environment. As described earlier this is not the case. Fortunately, there are more and more standards appearing in the latest grid middleware releases. As described in section 4.6.1, the gridFTP standard is already present in the middleware environments that are most important for GridAssist. And the WSRF, is implemented completely in GT4 and partially in the gLite middleware. According to developers of gLite, their middleware software will in the future be completely refactored towards WSRF standards.
Chapter 5

Architectural Assessment Methods

The decisions that are made during early phases of software engineering projects, such as during architectural design specifications, are most fundamental for the final software product. These decisions are documented in the software architectural design document. The presentation of this document should give stakeholders an idea about the software product’s conformance to their wishes and expectations. When a divergence between the requirements and the software architecture exists, the software product is found inappropriate. Mistakes and wrong design decisions are costly to fix after the software has been built. Therefore, it is important to identify them as early as possible.

5.1 Existing Methods

Ideally, one of the most useful assessment tools during software design would be one that can predict the quality of the software system from a high-level design description. This idea was first introduced by Parnas [47], who described modularization and information as a high-level system decomposition to improve flexibility and comprehensibility. Complete verification of a software design is unfortunately too complex and costly, but there exist rather useful methods for quality attribute assessment.

The requirements specification phase produces functional and quality requirements, which in its turn form the basis for the scenarios. Scenarios are a postulated set of uses of the system. A scenario describes the system’s expected reaction to user actions concerning a certain task. Scenarios are a good way of synthesizing individual interpretations of a software quality attribute into a common view. A scenario is
more concrete than the general software quality definition, and also more context sensitive (i.e. it incorporates a description of the system). Software architecture analysis tries to predict the quality (in respect to attributes such as availability, maintainability, security, etc) of a software system based on the descriptions of the architectural design. An architectural assessment links the quality attributes, in the form of scenarios, with the software architecture that is described in types, structures, decisions and system context. Basically, a software assessment tries to answer the question; how well does the software architecture supports the quality attributes?

A software architecture assessment can either be performed very early in the architectural design process or later on during the detailed design phase. An early assessment functions as a prediction of how well requirements will be met. When the assessment is performed in a later phase, it acts as a verification tool for the architecture being developed. Drawbacks of an early assessment can be incomplete design specification or even totally missing documentation. On the other hand, when an assessment is performed too late, there is a risk of having to redo a great deal of work [15]. The assessment of GridAssist is performed after the software architecture design phase, after which enough design information should be available and before the detailed design and implementation phases.

5.1.1 Quality Attributes

Software architecture assessment methods each have a particular scope on assessing some quality attributes. The choice for the most suitable assessment methods depends on the choice for the quality attributes that are to be assessed. In our case, this choice is based on the original requirements of GridAssist [49, 50], on the requirements of the adaptations to GridAssist (see appendix B) and on the expectations of GridAssist’s stakeholders. Firstly, GridAssist has a high score on the easiness of use and recommendations from other users showed that usability could be seen as the key attribute of GridAssist. Furthermore, because grids are evolving very rapidly and Dutch Space is continuously adapting GridAssist to follow the developments, the modifiability of GridAssist’s software architecture is also an important quality attribute. And thirdly, in order to keep the customer area as large as possible, the components of GridAssist need to be compatible with as many different hardware and operating systems as possible. Currently, the requirements give certain constraints on the hardware and operating system’s portability. Because this requirement influences the customer adoption it should be carefully watched for degradation.

The remainder of this chapter will briefly describe a subset of scenario based assessment methods that are focussed on either one or more of the quality attributes
discussed above (modifiability, portability and usability). There are many other useful assessment methods, but they are not applicable for this project. For a complete overview of assessment methods I refer to [19, 34]. In chapter 6 a selection will be made from the described methods for the assessment of GridAssist.

5.1.2 Software Architecture Assessment Method

The Software Architecture Assessment Method (SAAM) has been around since 1993, corresponding to the trend of better understanding of architectural concepts [34]. It was one of the first software architecture assessment methods and has formed the basis for many other methods. The goal of SAAM is to verify basic architectural assumptions and principles against the documents describing the desired properties of an application. Additionally, the analysis offers an inspection of the architecture focusing on potential trouble spots, such as requirement conflicts or incomplete design specifications from a particular stakeholder’s perspective. SAAM is stakeholder centric, which means that the scenarios brought forward by the stakeholders determine which qualities are investigated [37, 35]. For example, for modifiability assessment change scenarios are used and for a portability assessment porting scenarios.

The output of SAAM is a collection of per-scenario based metrics. These metrics can be combined to obtain a metric for the complete software architecture, or can be used to compare software architectures on a per-scenario basis.

SAAM consists of five steps, which are followed in a sequential order, however some steps may be repeated to refine the assessment. In the first step the candidate architecture is described, in a manner that is understood by all parties involved in the assessment. After that, the scenarios that are specific for a quality attribute are described by the stakeholders. These scenarios should capture all uses of the system and should therefore be provided by a representative group of stakeholders. In the third step, the scenarios are analyzed for architecture support to perform the scenario. When the scenario describes features of the system that are not supported by the architecture, the scenario is labeled indirect and if it is supported it is labeled direct. For each indirect scenario, the impact on the software architecture should be noted (for example a list of affected components). The next step looks at all the affected components and analyzes which scenarios interact by affecting the same component. These scenario interactions indicate into which extent a software architecture separates its concerns. The last step of SAAM allows an evaluation of the results of previous steps. It is possible to weigh each scenario and calculate an overall ranking of the architecture.
5.1.3 Architecture Trade-Off Analysis Method

The Architectural Trade-Off Analysis Method (ATAM) was proposed as an improvement of SAAM. The purpose of ATAM is to assess the consequences of architectural design decisions in light of quality attributes [19]. In order to satisfy the quality attributes in the best possible way, ATAM provides software architects with a method to choose between several architectural approaches. As with SAAM, ATAM also uses scenarios as foundation for identifying the properties of a software architecture [38, 39].

ATAM consists of four phases of which the most important ones (phase 1 and 2) can be divided into the nine steps described below [6]. During these steps, all of the stakeholders need to participate and provide input.

1. Presentation of ATAM to all the involved stakeholders. This step is to explain the context and expectations for the activities of ATAM.
2. Present an overview of the context of the application and the business drivers that are responsible for the development of the application.
3. The architect presents the architecture to the involved people. Enough detail should be provided, dependent on the available time, the quality requirements, and the completeness of the documentation.
4. Architectural approaches and patterns represent the alternatives that an architect has in developing an architecture. The approaches and patterns used in the architecture under investigation should be identified.
5. With the use of a utility tree, the quality goals of the software architecture are identified. Basically, the architecture’s influence on each quality attributes is identified.
6. Analyze the architecture for its support of each of the scenarios. During this activity the risks, sensitivity points (a property of one or more components that is critical for achieving a desired quality attribute) and trade-off are documented.
7. A brainstorm session is started with the goal to identify over-looked scenarios. Once all scenarios have been collected, they are prioritized.
8. The scenarios are analyzed according to their priority, the highest-ranking scenario will have the most influence on defining the importance of architectural approaches.
9. Present results the stakeholders.

ATAM outputs the architectural design decisions that satisfy the requirements in the best possible way. This should help software architects in making the rational choice when designing. Furthermore, ATAM can be used as an early clarification of the requirements.
5.1.4 Architecture Level Modifiability Analysis

Architecture Level Modifiability Analysis (ALMA) is an assessment method that focuses specifically on modifiability. This assessment method is a combination of SAAM and ATAM [40, 34], and therefore the steps taken in the assessments are alike to SAAM. The goals of ALMA’s modifiability analysis are maintenance effort estimation, risk assessment and a comparison of candidate architectures [9]. Depending on the chosen goal ALMA uses different techniques in the main steps. ALMA uses change scenarios to indicate the possible need for changes.

The following five steps are taken when ALMA is performed. The exact techniques used in these steps are determined by the goal determined in step 1.

1. Determine the goal of the analysis, which can be maintenance cost prediction, risk assessment or software architecture selection.
2. Describe the software architecture in views of the decomposition of the system in components the relationships between components, and the relationship with the system’s environment.
3. Elicit scenarios that can be used in the assessment. This involves identification of stakeholders, interviewing stakeholders, and documenting the results of these interviews.
4. Evaluate scenarios by performing an impact analysis, consisting of the following steps: identification of affected components, analysis of the effect on these components and the ripple-effect (i.e. changes required due to changed components) on other components.
5. Interpret the results and draw conclusions about the modifiability support of the software architecture.

5.1.5 Systematic Quantitative Analysis of Scenarios’ Heuristics

The Systematic Quantitative Analysis of Scenarios’ Heuristics (SQUASH) method is used for assessing the quality level expressed at a user scenario level. However, SQUASH has been particularly developed for the assessment of usability [33].

This method consists of seven steps, divided into two phases, namely information gathering and decision making. The first step involves the identification of stakeholders, who will provide information concerning the different quality objectives in the second step. The quality attributes have to be defined in a quantitative way, in order to be measurable and usable in the quantitative assessment. In step four, the scenarios are analyzed, to find out the conformance with the quality objectives. This results in a scenario quality profile that is to describe the conformance of the
scenarios with the quality attributes. In case this indicates that some scenarios are not acceptable for the stakeholders, an improvement of the bottleneck-scenarios is needed. In the final step a decision is made on the most preferred scenario from the quality objective’s point of view.

5.1.6 Scenario-Based Architecture Re-engineering

Scenario-Based Architecture Re-engineering (SBAR) estimates the potential of the designed architecture to reach the software quality requirements. This method can be used to verify and re-engineer the software architecture, until the majority of the scenarios for a quality attribute are satisfied by the design [19, 8]. SBAR is not really applicable for one quality attribute but more for assessing the software architecture as a whole.

SBAR uses four different approaches for software architecture evaluation:

- Scenario-based evaluation; as described in previous methods scenarios are useful for bringing understanding of what a particular software quality really means.
- Simulation of the software architecture by using an implementation of the architecture. Only the main components of the architecture are implemented, the other components are simulated. This setup can be used to simulated system behavior under various circumstances.
- Mathematical modeling can be used to statically evaluate operational software qualities.
- Experience-based reasoning, which uses experience for assessment on the qualities of the software architecture. Experienced software engineers often have valuable insights that may provide information concerning the assessment of quality attributes. Furthermore, the experience can be extremely helpful in avoiding bad design decisions. However, this approach is very subjective and uses tacit knowledge of the participants.

The input of SBAR consists of the updated requirements specification and the existing software architecture, as output an improved and re-engineered software architecture is constructed. This is done in four steps, beginning with inserting the new functional requirements in the architecture. This is followed by an assessment of each software quality attribute, according to the approaches described above. Based on the output of the assessment the architecture is transformed to better support the quality attributes. After the adaptations to the software architecture, the quality assessment is repeated until all quality attributes are conform the requirements.
5.2. An Additional Assessment Method

5.1.7 Scenario Based Architectural Level Usability Analysis Method

Scenario Based Architectural Level Usability Analysis (SALUTA) method has been specifically developed for the assessment of the usability aspect of software architectures [25]. This is done with a usability framework that describes usability patterns and properties of a software architecture (see figure 6.2). The presence of architectural sensitive usability patterns determines the architecture’s support for usability. The presence of architectural usability properties is used to identify the architecture and the design decisions that lead to this architecture for usability support. The quality of the SALUTA assessment depends on the amount of evidence for patterns and properties that can be extracted from the software architecture [26].

The SALUTA method consists of six steps, starting with the creation of usage profiles that help the assessor in describing the expected usability of the architecture in terms of satisfaction, learnability, efficiency and reliability. The second step is the description of the architectural usability, which is performed according to the usability framework that describes usability patterns and properties. The third step analyzes which usability properties and patterns the scenario is affected by. The fourth and fifth steps compare the provided usability (step 3) with the desired usability (step 1). Finally, the results are presented in the last step. For clarification the results are mapped onto four usability sub-attributes (learnability, reliability, efficiency and satisfaction).

5.1.8 Software Architecture Evaluation Model

Software Architecture Evaluation Model (SAEM) establishes the basis for the quality evaluation and prediction of the quality of the final system [19]. This method distinguishes itself from the other methods by assessing the quality of the whole system, where the other methods just look at one (or a few) quality attributes.

SAEM uses metrics that are based on the Goal Question Metric (GQM) technique, which means it assesses the software architecture according to a measurement model that consists of a conceptual level (goal), an operational level (question) and a quantitative level (metric). The GQM approach is a mechanism for defining and interpreting operational and measurable software [5].

5.2 An Additional Assessment Method

There are not so many assessment methods that assess the usability aspect of a software architecture. There is a logical explanation for this; the software architecture phase is not really concerned with designing the graphical or command line user
interfaces. This is often done in later stages of the software development process. Besides, the complexity of a user interface depends on many other factors, such as user experience and the scenarios a user uses. This makes the usability assessment during the software architecture phase very hard, and it cannot be completely separated from decisions made during later stages. Despite this consideration, it is important that information is gathered on how decisions made during architectural design influence usability. It is necessary that the software architecture is assessed for its usability support.

As discussed in the previous chapter, SALUTA provides one solution that focuses on architectural properties and patterns, which will influence the usability of the software application. The influence of the adaptations that are made to the architecture of GridAssist, on the complexity of the user interface is examined. When SALUTA is used in this case, it will not deliver any information on the screen complexity, but rather a view on the differences in usability patterns and properties. It can easily be seen that the patterns and properties are identical in the original and the adapted software architecture. SALUTA will only find a difference caused by the number of scenarios, which is obvious and not the goal of this assessment. The same can be noticed in the other method, called SQUASH. Therefore, a new method that especially focuses on how adaptations to the software architecture influence the complexity of the user interface. This is done following the usage scenarios that can be elicited from the software architecture design.

5.2.1 Scenario-based Architecture Comparison of Usability

One aspect of the usability assessment is the difference between the original and adapted software architecture. We are particularly interested in the number of affected scenarios and impact of these affected scenarios. Unfortunately, there are currently not many scenario-based usability assessment methods that can be applied during the architectural design phase. Therefore, I have retrofitted several known methods into a new Scenario-based Architecture Comparison of Usability (SACU) method. This method is to compare the usability of an original architecture with the adapted software architecture, based on the Graphical User Interface (GUI) complexity.

SACU is a method that compares GUI mock-ups from before and after software architecture adaptations to indicate the influence on the usability quality attribute. The scenarios indicate system usage and can be gathered in a similar way as is done in the SAAM method. SACU can be used in combination with SALUTA, to completely validate the usability of the adaptation to the software architecture. SALUTA will assess the absolute usability support of the architecture. SACU will
assess the architecture in relation to the original architecture to indicate degradation or improvement of usability.

When a change scenario involves the addition of a high number of windows and elements the user needs more time for performing the scenario. Furthermore, more windows and elements significantly increase the complexity of the scenario (and thereby also of the whole application). Despite the difficulties in predicting usability during architectural design phase, there are some relations between early design decisions and a usable system. Over an average user set (in experience, age and skills) who together use all functionality, it can be observed that the more windows and elements are involved in a scenario:

- The more time the user will require for performing the scenario.
- The more complex the scenario will become.
- The harder it becomes for the user to learn using the application.

In order to measure the complexity difference SACU follows the following five steps:

1. Identify scenarios that describe the functions that a user can either perform in the original and in the adapted architecture. In most cases, the latter is an extension of the former.
2. Analyze the original architecture for the number of windows and elements that are necessary for the user to perform the scenarios. The elements can be buttons, drop-down boxes, text fields or input fields. The sum of the number of windows and elements per scenario is called the complexity value.
3. The number windows and element and the complexity value per scenario are analyzed for the adapted architecture.
4. Comparison of the results of the previous two steps is done by calculating the difference between the complexity values between the original and the adapted software architecture. From these values the difference sum, the difference percentages, and the average changes are calculated.
5. Present results.

One of the methods that is related to SACU is McCabe’s cyclomatic complexity [42]. This method tries to investigate the complexity of a piece of software by looking at the number of decision statements in a program. These decisions are resembled by if statements in the code. SACU also tries to investigate the complexity of a program, but looks only at the user interface. Instead of program statements, SACU uses screen elements and windows as basis for its complexity investigation. The goal of McCabe’s complexity measure is to determine whether the program code is not too complex. Complex code is harder to understand, which increases the chance on
mistakes. Whereas SACU tries to investigate the user interface complexity. Both methods look at a program’s complexity, but from a different perspective.

The input entities of SACU consist of the scenarios and a description of both the original and the adapted software architecture. The software architecture must be specified in enough detail for the analyst to identify the number of windows and elements for each scenario. The results of SACU are the change percentage of screen complexity between the original and adapted architecture, the change percentage of affected scenarios, and the average screen complexity increase per scenario.

5.2.2 Reflection on SACU

SACU is a method that can be used to compare the usability of a program before and after adaptations have been done. The results of SACU tell something about the nature of the changes. An identification of the following facts can be obtained from SACU:

- The most complex scenarios from a usability point of view. It might be worth lowering the number of elements involved in these scenarios, in order to make the application less complex (i.e. more usable).
- The scenarios that get less complex (when for example the number of windows and elements decrease), can indicate that the goal of usability improvement have been met on the scenarios. Also an assessment over all scenarios is possible, when the goal is to decrease the total complexity of the whole application.
- The average complexity increase over all scenarios should give a good impression of how deep into the system the influences of the changes are penetrating.
- The number of elements and windows for the new scenarios will indicate the complexity of the changes in relation to the complexity of the original system. When the complexity of the added scenarios (i.e. added functionality) is far higher than the complexity of the original unchanged scenarios, SACU indicates that the changes are more complex than the system was originally meant to be.

SACU is not a method that should be used stand-alone to assess the usability of a software architecture. SACU has one specific goal and should be used only for that goal. Thimbleby [51] states that complexity is a valid, but not the only, measure for usability. Therefore SACU should be used in combination with other usability assessment methods (e.g. SALUTA, SQUASH) to make a more complete assessment of usability possible.

One point that is worth mentioning, is that during the software architectural design
the GUI might not be completely specified. During detailed design and implementation the exact layout of the GUI is determined. Only then also the exact complexity of the GUI can be determined. SACU determines the screen complexity changes between the original and adapted software architecture based on the number of windows and elements in each scenario. This relation will be uninfluenced by the detailed design and implementation phases. Besides, complexity is dependent of the user, and SACU only gives an average complexity indication for the average user. SACU will not predict how complex the system will be for all users, but rather where complexity complications are likely.
Eight different architecture assessment methods have been discussed in the previous chapter. The seven methods from section 5.1 represent the most commonly used scenario-based assessment techniques that are applicable for the quality attributes. Some methods were specifically developed for a single goal (i.e. one quality attribute) and some are used in wider areas than just architecture evaluation (e.g. cost estimation). The method discussed in section 5.2 has been specifically created for comparing the adapted GridAssist’s screen complexity with the original. In order to select the optimal method (or combination of methods) for evaluating GridAssist’s software architecture, a comparison between the nine methods is required.

It has been discussed in section 5.1.1 that modifiability, portability and usability are important for the success of GridAssist. The other quality requirements should not be disregarded and can also be recommended for an architecture assessment. But an assessment is very time-consuming and the scope of this project does not allow a full-scale assessment of the complete requirements set. Therefore, a selection of the most suitable assessment methods for only modifiability, portability and usability is necessary.

### 6.1 Method Applicability

The list below gives a description of the applicability of each method from the previous chapter. The description and applicability of all assessment methods has been summarized in table 6.1. Each method has been given a score indicating its applicability. "Poor" indicates that it is not applicable for our assessment, but can be used when no better method is available. "Medium" indicates that it can be used under some conditions. And "Good" indicate that the method is perfectly
suitable for our assessment. Based on this score, a choice will be made for the most applicable methods (or combination of methods).

**SAAM** can be used for assessing portability and modifiability. This method has known a long period of use and has evolved into a full-grown assessment method. Furthermore, this method is simple to use, in contrast to some of the methods that have been built on it.

**ATAM** has two goals, the assessment of architectures and making a trade-off between them. ATAM is quite complex, and by far the method with the most activities. This method identifies trade-off and sensitivity points to make a comparison between software architectures candidates possible, which is not really necessary in this assessment.

**ALMA** has three possible goals, of which the architecture comparison can be used in our assessment. Because this method is specifically designed for modifiability it will surely allow a trustworthy assessment.

**SQUASH** has been particularly developed for the assessment of usability. This method quantifies the factors that relate to usability and uses these quantified values to give a conclusion about the architecture’s usability support. This method can be used in the usability assessment of GridAssist, but quantification is not really applicable for all usability aspects of GridAssist.

**SBAR** and **SAEM** have both been developed to assess a software architecture as a whole. This is extremely useful, but a complete system assessment is out of the scope of this project. These assessment methods are therefore currently not applicable.

**SALUTA** assesses usability in an instinctive way, with the use of a usability framework of patterns and properties. This method can be used in the usability assessment of GridAssist.

**SACU** has been specifically developed for the assessment of screen complexity. It is applicable to be part of the usability assessment, because neither SALUTA or SQUASH look at the screen complexities.

The assessment of portability is only supported by SAAM, this method will therefore be used in our assessment. For the assessment of modifiability, both ALMA and SAAM are applicable. ALMA is specialized for modifiability assessment and is built on top of SAAM. In order to create an applicable architecture assessment (i.e. to minimize cost and schedule implications) it is not optimal to use separate methods for each attribute, because that would require additional time for learning,
6.1. Method Applicability

<table>
<thead>
<tr>
<th>Method</th>
<th>Subject</th>
<th>Applicability</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAAM</td>
<td>Multiple attributes</td>
<td>Simple, widely adopted and used. Applicable for multiple attributes.</td>
<td>Good</td>
</tr>
<tr>
<td>ATAM</td>
<td>Trade-off</td>
<td>Complex and also oriented towards sensitivity and trade-off points.</td>
<td>Medium</td>
</tr>
<tr>
<td>ALMA</td>
<td>Modifiability</td>
<td>Architecture comparison goal can be used.</td>
<td>Good</td>
</tr>
<tr>
<td>SQUASH</td>
<td>Usability</td>
<td>Quantifies the factors that relate to usability, which is not always possible for GridAssist.</td>
<td>Medium</td>
</tr>
<tr>
<td>SBR</td>
<td>Re-engineering</td>
<td>Focuses on whole system, which is out-of-scope for this project.</td>
<td>Poor</td>
</tr>
<tr>
<td>SALUTA</td>
<td>Usability</td>
<td>Identifies patterns and properties, which is a possibility for GridAssist.</td>
<td>Good</td>
</tr>
<tr>
<td>SAEM</td>
<td>System quality</td>
<td>Focuses on whole system, which is out-of-scope for this project.</td>
<td>Poor</td>
</tr>
<tr>
<td>SACU</td>
<td>Screen complexity</td>
<td>Comparison of complexity between original and adapted architecture.</td>
<td>Good</td>
</tr>
</tbody>
</table>

Table 6.1 — Assessment Method Applicability

adapting and setting up the method. Since SAAM supports multiple quality attributes, is better documented and is easier (simpler) to use, it is more appropriate in this project for assessing both portability and modifiability.

For the assessment of usability we can use three methods, SQUASH, SALUTA and SACU. SALUTA and SQUASH both assess the architecture’s conformance with required usability, but each with a different approach. SALUTA uses an usability framework and SQUASH uses a quantification of usability related factors. This quantification process seems less reliable, because we will be unable to fully quantify all factors related to the usability of GridAssist. For example, it is hard to quantify the user satisfaction of GridAssist without a large user set. SALUTA on the other hand, would assess user satisfaction with the usability framework and indicate architectural patterns and properties that improve satisfaction. Therefore, SALUTA provides a better assessment and is therefore more applicable.

Because the complexity of user tasks is a relevant aspect of usability, SACU should be used as an additional method for the assessment of usability. SACU was constructed for comparing the complexity of the GUI according to scenarios that describe the functionality of the original and the adapted architecture.
6.2 Assessment Setup

Figure 6.1 shows the setup of the software architecture assessment of GridAssist. There are three quality attributes going to be evaluated with three different methods. The inputs of these methods consist of the original and the adapted software architecture of GridAssist. Besides that, relevant scenarios are needed as input for SAAM, SACU and SALUTA. Because SACU makes a comparison between the original and the adapted software architecture it needs the description of both software architectures as input. SALUTA is used to derive the difference between the desired and provided usability support, not to compare two software architectures (that where SACU comes in). That’s why this method uses only the adapted software architecture as input. SALUTA uses scenarios to evaluate the adapted software architecture usability score, and compares it with the desired usability score.

SAAM is performed four times, once for the original architecture to assess portability and once for the adapted software architecture to assess portability. Thereafter, SAAM is performed again in the same way to assess modifiability. The two outcomes (of the original and the adapted software architecture) for either portability and modifiability, are compared to indicate the difference in architectural support for modifiability and portability.
6.3 Modifiability Assessment

The results of this set-up will inform us of the difference in portability, modifiability, and usability support of GridAssist’s original and adapted architecture. Any architectural degradation due to the adaptations made, will be noticeable. The assessment of modifiability was performed with SAAM, the results are discussed in section 6.3. Portability is also assessed by SAAM and is discussed in section 6.4. And finally, section 6.5 will describe the assessment of usability with SALUTA and SACU.

6.3 Modifiability Assessment

This section presents the results of the modifiability software architecture assessment, that was performed with SAAM.

SAAM was performed on specific change-scenarios that describe the need and nature of changes that might occur during the operation of the software system. The change-scenarios form the basis for the assessment of the architecture. For each of these scenarios, the required changes in both the legacy and the adapted software architecture are given. Table 6.2 provides an overview of the differences in modifiability between the software architectures.

The directness as indicated in the two middle columns of table 6.2 indicate whether the change scenario can be handled (direct) by the current software architecture, or that modification is necessary (indirect). Some scenarios are not supported by the legacy software architecture, in which case modifications would require tremendous changes to the legacy software architecture. Therefore, these scenarios are not considered useful for comparison of the two architectures and labeled with a dash. For each indirectly supported scenario, table 6.2 also lists the number of components that are changed by it.

The interactions between change-scenarios are of great importance for our modifiability assessment. Components that are changed by more than one scenario are a potential modifiability concern, because these components can be changed for different objectives. This can cause complications when these objectives do not match. The result of the scenario interaction analysis for modifiability is documented in table 6.3, where the second and third column indicate the number of scenarios that affect the specific component. The last column indicates the interactions difference between the original and adapted software architecture. When the adapted software architecture has less interactions it has a better separation of modifiability and more it is able to support modifiability in relation to the legacy architecture.
Table 6.2 — SAAM modifiability assessment

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Legacy</th>
<th>New</th>
<th>Architectural Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add support for a new middleware environment into GridAssist</td>
<td>-</td>
<td>Direct</td>
<td>None</td>
</tr>
</tbody>
</table>
| Change CoG kit version for one middleware environment                     | Indirect (2) | Indirect (1) | **Legacy:** Change workflowtool for credential generation, and the GlobusWrapper for communication.  
**Adapted:** Change the specific wrapper implementation that uses the CoG kit. |
| Adapting the system to cope with changed certificates of an already supported middleware environment | Indirect (2) | Indirect (2) | **Legacy:** Change the Proxy class that represents the Credential. And change the generation of the proxy in the workflow tool. If the required input of the user, for this certificate, is also changed, then the Workflow Tool component needs to be adapted.  
**Adapted:** The wrapper implementation of the specific middleware type needs to be changed to cope with the changed certificate. If the required input of the user, for this certificate, is also changed, then the Workflow Tool component needs to be adapted. |
| Add resource                                                              | Direct | Direct | None                  |
| Add service                                                              | Direct | Direct | None                  |
| Add certificate for supported middleware type                            | Indirect (2) | Indirect (2) | **Adapted and Legacy:** The WorkflowEngine should be able to act as an agent (sends new releases) and the Controller as a dispatcher (acts on a release). |
| Add support for Notification system (NoteCaster-Express)                 | Indirect (2) | Indirect (1) | **Legacy:** Change WorkflowTool for credential generation, and the GlobusWrapper for communication.  
**Adapted:** Change the specific Wrapper implementation that uses the CoG kit. |

It can be concluded from table 6.3 that there is no drastic decrease in the support for modifiability due to the adaptations to GridAssist. In the legacy software architecture, the workflowtool component is more frequently undergoing modifications, whereas in the adapted architecture, more adaptations to the Wrapper component are needed for some scenarios. The sum of all interactions shows that fewer modifications are needed in the adapted architecture. This means that the adapted software architecture has a better separation of functionality and is therefore easier to modify.

### 6.4 Portability Assessment

This section discusses the portability assessment that was performed with the SAAM method. This method has been performed on porting scenarios, which describe the
### 6.4. Portability Assessment

<table>
<thead>
<tr>
<th>Component</th>
<th>Legacy</th>
<th>New</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credential</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Wrapper</td>
<td>2</td>
<td>3</td>
<td>New software architecture needs 1.5 as much changes to this component.</td>
</tr>
<tr>
<td>WrapperLoader</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Controller</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>WorkflowEngine</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Permission</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Workflow Tool</td>
<td>3</td>
<td>0</td>
<td>New software architecture needs no changes to this component.</td>
</tr>
<tr>
<td>Monitor Pane</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Registry Pane</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Workflow Pane</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3 — Scenario interaction by module

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Legacy</th>
<th>New</th>
<th>Architectural Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port WorkflowTool to different operating system</td>
<td>Direct</td>
<td>Direct</td>
<td>None</td>
</tr>
<tr>
<td>Port Controller to different operating system</td>
<td>Direct</td>
<td>Direct</td>
<td>None</td>
</tr>
<tr>
<td>Port the server (Controller and Middleware wrapper) to different (older) Java version</td>
<td>Indirect</td>
<td>Indirect</td>
<td>GridAssist will run on JRE 1.3 without any problems, but compatibility with JRE 1.2 might require some minor modifications.</td>
</tr>
<tr>
<td>Port the WorkflowTool to a different (older) Java version</td>
<td>Indirect</td>
<td>Indirect</td>
<td>GridAssist will run without any troubles on JRE 1.3, but compatibility with JRE 1.2 might require some minor modifications.</td>
</tr>
</tbody>
</table>

Table 6.4 — SAAM portability assessment

need and nature of the porting of components that might occur during the operation of the software system. These scenarios form the basis for the portability assessment of the architecture.

As can be seen in table 6.4, the changes to the software architecture do not have any noticeable effect on the portability of GridAssist; the porting changes are the same for the legacy and the adapted software architecture. This seems logical, because neither the operating environment nor the programming language requirements are changed by the adaptations. Therefore it is not necessary to continue the portability assessment with SAAM beyond this point. Portability is not affected by the adaptations made to the software architecture.
6.5 Screen Complexity Assessment

Usability was assessed with both SACU and SALUTA. The former for screen complexity and the latter for software architecture’s usability support. The combination of these two methods provides an assessment of usability, in both an absolute way and in relation with the original software architecture of GridAssist. This section describes the results of the screen complexity assessment.

SACU was performed on the original and the adapted scenarios of GridAssist. Table 6.5 lists in the second and fifth column the number of windows \( W^\alpha \) and \( W^\beta \) and in the third and seventh column the elements \( E^\alpha \) and \( E^\beta \). These elements represent input boxes and lists that the user encounters when performing a specific scenario. The \( \alpha \) indicates the value for the original software architecture and \( \beta \) indicates the value for the adapted software architecture. The elements represent the number of choices that a user can make over all windows (for the whole scenario). The index value \( (\Sigma^\alpha \text{ and } \Sigma^\beta) \) is the summation of the number of windows and the number of elements for each scenario. The index deviation indicates the difference in the index values of the original and the adapted scenarios \( |\Sigma^\alpha - \Sigma^\beta| \). The index deviation per scenario is a good indication for the range and size of the complexity difference. The last column lists whether a scenario is either new (scenario is not present in original architecture), changed (number of windows and elements has changed), or unchanged (number of windows and elements in scenario are unchanged by adaptations to the architecture).

This approach was chosen because most users get disorientated when they have to work themselves through multiple windows. When a user needs to fill in a high number of elements, he or she can get distracted. Furthermore, the user interface might be hard to oversee and will therefore be (for the average user) harder to learn. The number of windows and elements was identified from the requirements of the architectural design and based on the user interface of the original GridAssist.

The most basic conclusion about the screen complexity that can be read from table 6.5 is the average windows and elements per scenario. In the original situation there were 18 scenarios, with a sum of \( \Sigma^\alpha \) of 90. This means an average of 5 elements/windows per scenario. In the adapted situation there are 23 scenarios and the sum of \( \Sigma^\beta \) equals 112, which results in 5.3 elements/windows per scenario. This indicates that the two versions are closely alike in respect to the number of windows and elements per scenario. In other words, the average complexity per scenario has not increased.

There are 23 scenarios in GridAssist, of which 9 have been modified (are either
6.5. Screen Complexity Assessment

### Table 6.5 — SACU results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Original</th>
<th>Adapted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W</strong>^α <strong>E</strong>^α <strong>Σ</strong>^α</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>W</strong>^β <strong>E</strong>^β <strong>Σ</strong>^β</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>**</td>
<td></td>
<td><strong>Σ</strong>^α − <strong>Σ</strong>^β</td>
<td></td>
</tr>
<tr>
<td>Add resource</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Change resource</td>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>View resource</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Add middleware type</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Remove middleware type</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Submit workflow</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Obtain certificate</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Add certificate</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Change certificate</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Delete certificate</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Job monitoring</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>File upload</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>File download</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Add service</td>
<td>3</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>Change service</td>
<td>3</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>Remove service</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Construct workflow</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Save workflow</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Load workflow</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Stop job</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Remove job</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Remove file</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>View storage</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>90</strong></td>
<td><strong>112</strong></td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>

The sum of all differences is 22, which roughly equals one element or window change per scenario. In other words, the average number of visual changes over all scenarios for GridAssist is 0.96 (\(\frac{22}{23}\)). Furthermore, we can conclude from this assessment that there are 2.44 (\(\frac{22}{9}\)) visual changes (both window and element) per changed scenario.

### 6.5.1 Interpretation of Results

Unfortunately, there is not much practical data available on the meaning of these values. How much changes in windows and elements a user can handle, before the...
experienced usability drastically decreases, is a psychological research area. Already in 1956 Miller has published about an investigation to how much information a human can handle [43]. It seems that it was an evolutionary advantage to be responsive to many inputs, but not in much detail. As Miller brings it: "The point seems to be that, as we add more variables to the display, we increase the total capacity, but we decrease the accuracy for any particular variable. In other words, we can make relatively crude judgments of several things simultaneously".

Miller introduces the term *span of absolute judgment*, which indicates that we are limited in the accuracy with which we can identify multiple information inputs. The span of absolute judgement is approximately seven. This indicates that the difference between the number of windows and elements in scenarios of the original and adapted version of GridAssist is not important as long as the span of absolute judgement is not exceeded. Apparently, the complexity of the scenarios in GridAssist are not drastically high (i.e. within limits). The consequences for usability will probably not be dramatic.

6.5.2 User Walkthrough

In order to be able to give further explanation of the results of SACU, I performed a small walkthrough analysis with three users. The walkthrough should also give insight in how the overall usability of GridAssist is experienced by users. This walkthrough has been performed after the implementation and testing phase, whereas the software architecture assessments (e.g. SACU) have been performed before detailed design. The goal was therefore not to predict usability of the system, but rather to verify the results of the assessments. The walkthrough has been based on a user walkthrough performed by TNO [44].

All users were new to GridAssist, which made it possible to observe how intuitive the user interface of GridAssist is. The walkthrough consisted of a number of tasks that the user needed to perform, first on the original version of GridAssist and thereafter on the adapted version (see appendix D). After performing the tasks, the user could give his opinion on the usability and complexity difference. Furthermore, I observed the user’s behavior and the easiness with which the tasks were performed.

The walkthrough indicated that the users who are new to GridAssist can, with minor explanation, easily work with it. The user interface has a very intuitive layout, and a very short learning period. The users indicated that the adaptations were in the same style as the original version, which made it very easy for them to work
with the adapted version of GridAssist. The perceived complexity increase due to the adaptations to GridAssist is therefore very small. Another important result of the walkthrough is that the adaptations are mainly concerned with administration functions. Normal usage of GridAssist is not concerned with managing resources, services and middleware types. The walkthrough indicated that the adaptations did not influence the normal user, but rather the administrator, who is better trained in using GridAssist and has more experience to handle complex scenarios.

6.6 Usability Assessment

This section describes the six steps of SALUTA and presents the results of the steps taken. These steps are based on [25, 26], but some changes were made to fit into our project.

Step 1: create usage profile
SALUTA divides usability into four categories:

- Satisfaction; the user should feel that he is in control.
- Learnability; how easy is this task for the user to understand.
- Efficiency; the time it takes the user to perform this task.
- Reliability; how error-free this task can be performed.

The table 6.6 below shows the usage profile of GridAssist, which indicates the importance of each of the usability quality sub-attributes. These numbers can be seen as usability requirements, the minimal acceptable score, in the optimal situation they are as high as possible. This information was gathered in stakeholder interviews, and extracted from three usage types. Each usage type expects other scores on the usability attributes:

- Normal usage includes tasks that every user can or needs to do, for these tasks reliability, and in less extent learnability, are important.
- Administration usage includes tasks that update the registry. The administrator wants to do his tasks as fast as possible, that is why efficiency is important. Furthermore, an administrator needs to feel that he/she is in control, therefore also satisfaction is important for these tasks.
- Functional usage are the tasks that are made possible by the other tasks, these tasks provide the main functionality of GridAssist and are performed by
non-experts, therefore learnability and reliability are important. Because the user needs to be in control of the data processing tasks, satisfaction is also important here.

The goal of performing an assessment is not to evaluate all scenarios, but only a representative subset [26]. This assessment is used to analyze the usability requirements conformance of the adaptations, therefore a scenario set that is representative for the adaptations is used. The scenarios typically describe adapted functionality or were affected by changes to the architecture. For each scenario the importance of satisfaction, learnability, efficiency and reliability is given according to the usage type of the scenario (see table 6.6).

### Step 2: describe architectural usability

In this step information about the software architecture is collected. Two types of analysis techniques are used; usability pattern based analysis and usability property based analysis. These two elements form a usability framework that defines the relationship between the usability and the software architecture [25] (see figure 6.2).

The usability patterns and properties from the usability framework [26] are listed and analyzed in tables 6.7 and 6.8. These tables also list whether the patterns and properties are present in (i.e. supported by) the adapted architecture of GridAssist. This analysis has shown that the architectural usability is identical for the original and the adapted architecture of GridAssist, because both architectures support the same usability patterns and properties. We continue the SALUTA assessment with only the new software architecture, in order to see if the scenarios support the usability quality attributes as is desired.
6.6. Usability Assessment

Figure 6.2 — SALUTA’s usability framework

Step 3: identify patterns and properties
In this step for each scenario it is analyzed by which usability patterns and properties (tables 6.7 and 6.8) it is affected. This is done by identifying which properties and patterns are related to each scenario according to the values given in table 6.6. For example, the usability of scenario ‘add resource’ is affected by the patterns ‘context sensitive help’, ‘data validation’ and ‘system feedback’. The results of this step are listed in table 6.9.
Table 6.7 — SALUTA usability patterns

<table>
<thead>
<tr>
<th>#</th>
<th>Pattern</th>
<th>Description</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Workflow model</td>
<td>The workflow model is used as an understandable representation of jobs.</td>
<td>Yes</td>
</tr>
<tr>
<td>P2</td>
<td>Scripting</td>
<td>Provide a mechanism that allows the user to perform a sequence of commands or actions to a number of different objects.</td>
<td>No</td>
</tr>
<tr>
<td>P3</td>
<td>User Modes</td>
<td>Provide different modes corresponding to different feature-sets required by different types of users, or by the same user when performing different tasks.</td>
<td>Yes</td>
</tr>
<tr>
<td>P4</td>
<td>Wizard</td>
<td>Present the user with a structured sequence of steps for carrying out a task and guide them through one by one. The task as a whole is separated into a series of more manageable subtasks. At any time, the user can go back and change steps in the process.</td>
<td>No</td>
</tr>
<tr>
<td>P5</td>
<td>User Profile</td>
<td>Build and records a profile of each type of user, so that specific attributes of the system (for example, the layout of the user interface, the amount of data or options to show) can be set and reset each time for a different user. Different users may have different roles, and require different things from the software.</td>
<td>No</td>
</tr>
<tr>
<td>P6</td>
<td>Context Sensitive Help</td>
<td>Monitor what the user is currently doing and make documentation available that is relevant to the completion of that task.</td>
<td>Yes</td>
</tr>
<tr>
<td>P7</td>
<td>History Logging</td>
<td>Record a log of the actions of the user (and possibly the system) to be able to look back over what was done.</td>
<td>No</td>
</tr>
<tr>
<td>P8</td>
<td>Data Validation</td>
<td>Verify whether (multiple) items of data in a form or field have been entered correctly.</td>
<td>Yes</td>
</tr>
<tr>
<td>P9</td>
<td>Cancel</td>
<td>Allow the user to cancel a command that has been issued but not yet completed, to prevent reaching an error state.</td>
<td>No</td>
</tr>
<tr>
<td>P10</td>
<td>Undo</td>
<td>Allow the user to call back a previous command and the system will restore to a previous state.</td>
<td>No</td>
</tr>
<tr>
<td>P11</td>
<td>Actions for Multiple Objects</td>
<td>Some tasks can be performed on all selected items (e.g. stopping jobs, removing jobs).</td>
<td>Yes</td>
</tr>
<tr>
<td>P12</td>
<td>System feedback</td>
<td>When needed, the system provides the user with additional information. For example, when some action cannot be performed.</td>
<td>Yes</td>
</tr>
<tr>
<td>P13</td>
<td>Multiple Views</td>
<td>Provide multiple views for different users and uses.</td>
<td>No</td>
</tr>
<tr>
<td>P14</td>
<td>Emulation</td>
<td>Emulate the appearance and/or behavior of a different system.</td>
<td>No</td>
</tr>
<tr>
<td>P15</td>
<td>Multi Channelling</td>
<td>Provide a mechanism that allows access using different types of devices (input/output).</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Step 4: identify number of references
This step translates the patterns and properties of each scenario (table 6.9) into an
Table 6.8 — SALUTA usability properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Mapping</td>
<td>Includes predictability of operation, semiotic significance of symbols and ease of navigation</td>
<td>No</td>
</tr>
<tr>
<td>Minimize Cognitive Load</td>
<td>System design should recognize human cognitive limitations, short-term memory</td>
<td>Yes</td>
</tr>
<tr>
<td>Adaptable</td>
<td>The system should be able to satisfy the user's needs when the context changes or adapt to changes in the user. Such property might be decomposed as follows: ability to adapt to changes in the user's level of experience; ability to provide certain customized services, capacity of the system for remembering past details of the user-system interaction</td>
<td>No</td>
</tr>
<tr>
<td>Guidance</td>
<td>On-line guidance as to the operation of the system</td>
<td>No</td>
</tr>
<tr>
<td>Error Management</td>
<td>Includes error prevention and recovery</td>
<td>No</td>
</tr>
<tr>
<td>Explicit User Control</td>
<td>Direct manipulation should be supported; e.g. the user should get the impression that he is &quot;in control&quot; of the application</td>
<td>Yes</td>
</tr>
<tr>
<td>Provide Feedback</td>
<td>The system provides continuous feedback as to system operation to the user</td>
<td>Yes</td>
</tr>
<tr>
<td>Consistency</td>
<td>Consistency of both the user interface and functional operation of the system</td>
<td>Yes</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Includes multi-mode access, internationalization and support for disabled users</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6.9 — SALUTA scenario’s patterns (α) and properties (β)

overview of the provided usability. For each scenario, the number of references to efficiency (E), reliability (R), satisfaction (S) and learnability (L) in the framework are listed. This is done for the patterns in column α and for the properties in column β, these values are summed for each usability attributes in column Σ. It can be seen in table 6.10 that satisfaction is overall the most important quality attribute, followed by reliability, efficiency and learnability.
### Table 6.10 — SALUTA references to patterns ($\alpha$) and properties ($\beta$)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Efficiency</th>
<th>Reliability</th>
<th>Satisfaction</th>
<th>Learnability</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Resource</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Change Resource</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>View Resource</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Add Middleware Type</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Remove Middleware Type</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Submit Workflow</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Obtain Certificate</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Add Certificate</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Change Certificate</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Delete Certificate</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>23</strong></td>
<td><strong>26</strong></td>
<td><strong>28</strong></td>
<td><strong>22</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.11 — SALUTA provided ($\gamma$) and desired ($\delta$) usability comparison

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Efficiency</th>
<th>Reliability</th>
<th>Satisfaction</th>
<th>Learnability</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Resource</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Change Resource</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>View Resource</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Add Middleware Type</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Remove Middleware Type</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Submit Workflow</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Obtain Certificate</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Add Certificate</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Change Certificate</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>Delete Certificate</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td><strong>Supported scenarios</strong></td>
<td><strong>10</strong></td>
<td><strong>10</strong></td>
<td><strong>9</strong></td>
<td><strong>8</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Step 5: compare provided and desired usability

This step compares the desired usability (table 6.6) with the provided usability (table 6.10). The provided usability is mapped to a 1-to-4 ranking; the highest score is mapped to a 4 and the second highest to a 3, etc. Identical values are mapped to the same value in the 1-to-4 ranking. For example 2-3-3-3 is mapped to 3-4-4-4. In this way a scaling is obtained, which can be compared with the desired usability values from table 6.6.

The results of this comparison are displayed in table 6.11. The comparison score (last column) shows how well the desired usability is met by the software architecture for each scenario. This value was obtained by comparing the desired and provided scores of the usability attributes. When an attribute’s provided score is at least the
desired score, the scenario’s measure gets a number of points equal to the desirability of the usability attribute. So, if a scenario’s provided usability score exactly matches the desired score it gets 10 points (4 for the most desirable attribute, 3 for the second most desirable attribute, etc). This scoring mechanism was chosen to make a fair comparison, because the highest provided usability attribute is always mapped to a 4, even when there are multiple identical values. This makes the comparison with the highest ranking desired usability attribute easier. Furthermore, the conformance on the highest ranking usability attribute has the highest importance and is therefore rewarded with the highest score. This calculation of the conformance percentages can be described with the following formula:

\[10 \cdot \sum_{i=1}^{4} i \cdot (\gamma_i \geq \delta_i)\]

Where \(\gamma_i\) stands for the provided support and \(\delta_i\) stands for the desired support, for either the highest rated desired usability attribute \((i = 4)\), the second highest \((i = 3)\), the second lowest \((i = 2)\) and the lowest \((i = 1)\). The boolean expression \(\gamma_i \geq \delta_i\) equals one when true and zero when false.

The average of all the percentages in the last column of table 6.11 yields an average usability conformance of 91%. This is based on the adapted scenarios and architecture of GridAssist. It can be seen that scenarios 'View resource', 'Add certificate' and 'Change certificate' are not fully supported by the software architecture. This does not mean that these scenarios are not usable, but rather that they have a lower conformance to what is expected from the score on efficiency, reliability, satisfaction and learnability. These scenarios are not less usable but just have another focus on the usability attributes (i.e. the focus is more towards efficiency and less on learnability).

**Step 6: present results**

The focus of the percentages in the last column is on the scenario’s conformance with the desired attributes. In order to retrieve understandable results out of the numbers in table 6.11, we need to compare the usability support attribute-based and not scenario-base, because we are not interested in the usability of each single scenario, but rather in the overall usability score. The following formula calculates the attribute’s conformance with the desired attributes:

\[10 \cdot \sum_{i=1}^{10} \gamma_i \geq \delta_i\]
where $\gamma_i$ stands for the provided attribute support for the $i^{th}$ scenario and $\delta_i$ stands for the desired attribute support for the $i^{th}$ scenario. The boolean expression $\gamma_i \geq \delta_i$ equals one when true and zero when false.

The usability attribute’s support in the software architecture is listed in table 6.12, 100% means that all scenarios provide the desired level of usability. It can be seen that all the scenarios match the desired efficiency and reliability, but satisfaction and learnability are only supported for about two thirds of the scenarios. Satisfaction and learnability might need some architectural improvements, although these scores are not critically low. Overall, the provided usability is lowest for learnability, but within acceptable limits.

**6.7 Conclusions**

Now that the assessments of modifiability, portability and usability have been performed, the results need interpretation. Basically, the software architect needs to know whether his designs are satisfactorily fulfilling the requirements. This section will first discuss the results of each of the quality attributes. After that, a general conclusion will be given for the whole software architecture.

**6.7.1 Modifiability**

SAAM was used to assess the difference in the support of modifiability of the two software architectures. There are some fundamental differences between the effect of the change scenarios on the legacy architecture and the effect on the adapted software architectures. In the legacy software architecture there was in some change scenarios a necessity to change the workflowtool component. This is extremely difficult because this component is distributed over the users’ computers. It is much easier to restrict modifications to one central location. In the adapted software architecture the architectural modifications are mostly concentrated to the wrapper component. Therefore, the adapted software architecture has a higher score on modifiability.
6.7.2 Portability

SAAM was also used to assess the difference of the support for the portability of the software architectures. The goal was to gather information about the portability difference between the legacy and the adapted software architecture. The assessment has shown that the portability of the software architecture is not affected by the adaptations done to GridAssist for grid middleware interoperability. Portability support after the adaptations is the same as it was before.

6.7.3 Usability

Usability was assessed with two methods; SALUTA was used to find out how well the adapted architecture meets the desired usability requirement, and SACU was used to indicate the relation in screen complexity between the original and adapted architecture.

SALUTA has shown that the adapted software architecture provides roughly the desired level of usability (see table 6.12). Usability can be divided into four sub-attributes; satisfaction, learnability, efficiency and reliability. Efficiency and reliability are exactly as desired, but satisfaction and learnability are only meeting the requirements in four out of five scenarios. The overall desired support is mainly focused on learnability and satisfaction and the main focus of the provided software architecture is more on user efficiency and reliability. This is a deviation between desired and provided usability. A possible improvement is the addition of usability properties and patterns from the framework that increase the software architecture’s satisfaction and learnability. Learnability should be improved for scenarios ‘add certificate’ and ‘change certificate’, for example by adding patterns like ‘multichanneling’ or ‘multiple views’. Satisfaction needs to be improved for scenario ‘view resource’ by adding patterns like ‘cancel’, ‘undo’ or ‘user profile’.

The results of the SACU method have shown that screen complexity increases in 39% of all scenarios. This increase was expected, because the addition of functionality involves adding screen elements. SACU analyzes this increase in elements in relation with the complexity of the original GridAssist. Furthermore, SACU analyzes how this increase in elements is distributed over the scenarios. The average element difference between the original and the adapted GridAssist is only 0.3. This means that the user will notice only a small overall complexity increase. There are 23 scenarios of which 4 (18%) have been changed, but each with only a relatively small complexity increase. Furthermore, there are 5 (22%) new scenarios that deal with the adaptations of the architecture. The user interface complexity increases with an average of one element per scenario, and about 2.44 (\(\frac{22}{9}\)) elements per changed...
Chapter 6. Architecture Assessment Process

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Results</th>
<th>Support</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiability</td>
<td>Changes shift from workflowtool to controller component</td>
<td>Complete</td>
<td>None</td>
</tr>
<tr>
<td>Portability</td>
<td>Not influenced by adaptations</td>
<td>Complete</td>
<td>None</td>
</tr>
<tr>
<td>Usability</td>
<td>Learnability and satisfaction not fully supported</td>
<td>Partly</td>
<td>Add architectural usability patterns</td>
</tr>
<tr>
<td>Usability (screen complexity)</td>
<td>Wide spread in influenced scenarios, but each with small complexity increase</td>
<td>Sufficient</td>
<td>Hide functions behind &quot;advanced&quot; buttons</td>
</tr>
</tbody>
</table>

Table 6.13 — Assessment results

scenario. The percentage of affected scenarios might be rather high, the changes are relatively small.

As was described in section 6.5.1, these values fall within the limits of the human mind. Also the walkthrough has shown that there is no dramatic complexity increase in GridAssist’s user interface. Is there anything that can be done about these numbers? It is unfortunately impossible to decrease the number of new scenarios, because each scenario itself is described by the user requirements, which are fixed. What can be altered is the number of windows and elements that are associated with a scenario. One possibility is to hide expert features behind an additional button, in order to decrease the number of elements that are associated with the non-expert use of a scenario. Furthermore, the influence of the added functionality on the original scenarios should be minimized. There are 4 changed scenarios, of which one gets less complex. The other three scenarios get more complex, but deal with functionality described in the user requirements (UR08 and UR09 in table B).

6.7.4 General

This section will give a small summary on the scenario-based assessment, for a very brief overview see table 6.13. Overall, it can be said that the quality attributes are not affected by the adaptations in a considerable manner. However, an architecture evaluation does not tell us "yes" or "no", "good" or "bad" or "70%". It rather tells us where we are at risk [15]. It can be seen that the risk is not in the portability and modifiability of the program. Usability on the other hand, does need some attention, because screen complexity has increased and the software architecture should increase its support for learnability and satisfaction.

With current requirements it is extremely difficult to reduce the screen complexity, as input is needed from the user. One way to accomplish lower screen complexities is to hide certain expert features for novice users. Furthermore, the architectural
support for usability should be improvement by the adding usability properties and patterns from the framework (see figure 6.2). Learnability should be improved for scenarios 'add certificate' and 'change certificate', for example by adding patterns like 'multi-channeling' or 'multiple views'. Satisfaction needs to be improved for scenario 'view resource' by adding patterns like 'cancel', 'undo' or 'user profile'. This will increase the architecture’s usability conformance with the desired usability support.
Related Work

Both software architecture assessment and grid computing are popular topics for academic research. This chapter describes the most important work, projects and literature that is related to this thesis. We start with a discussion about grid computing research, which is followed by a description of literature related to software architecture assessment.

7.1 Grid Computing

This thesis describes results on realizing interoperability of GridAssist between multiple grid middleware environments. The people from GridLab have developed a toolkit that can also be used to build interoperable grid client applications [3]. This toolkit, called the Grid Application Toolkit (GAT), provides developers of grid applications with a uniform programming interface. Just as the middleware framework in GridAssist’s design for interoperability, does this toolkit hide the complexities and the differences of the grid middleware environments. Because GAT also provides high-level functionality, it simplifies the creation of grid-aware application (i.e. a application that makes use of a grid’s functionality). At the start of this project this toolkit was still under development and support for the latest grid middleware environments was not available.

The CoG toolkit was mentioned earlier, it is a toolkit that can be used as a Java API for multiple grid middleware environments [41]. This toolkit is used in the wrapper that handles communication with the second version of the Globus toolkit. When the other grid middleware environments are supported, it could be used for other wrappers. The development team of CoG kit is currently working on support for the fourth version of the Globus toolkit, which is related to GridAssist’s GT4

1http://www.gridlab.org
wrapper. The difference between CoG and GAT is that the former is to provide an interface to Globus-specific, low-level grid interfaces, while GAT is more focused on providing higher level abstractions (e.g. file movement independent of underlying grid middleware).

The difference between these two toolkits and the work presented in this paper is the level of customization. The work on GridAssist is custom-made for its requirements. The toolkits are made for general use and provide general functionality for all kinds of grid-aware applications. However, there are many similarities between the work described here and that of the toolkits.

### 7.2 Software Architecture Assessment

The area of software architecture assessment is flooded with methods for assessing all kinds of possible aspects of a software architecture. In this thesis I discussed methods that are applicable for the important quality attributes of GridAssist. From these methods, three were chosen and used in the assessment. There are however many other interesting methods, each allowing interesting analysis of a software architecture.

Kazman, Asundi and Klein describe the Cost Benefit Analysis Method (CBAM), which bridges two domains in software development, namely the architecting process and the economics of an organization [36, 34, 4]. The methods discussed in chapter 5 focus primarily on quality assessment, this method looks at the economical aspects of a software architecture. CBAM sees costs, benefits and schedule implications as quality attributes of a software architecture and analyzes them. This method can be used as a follow-up of the architecture evaluation pre-session of ATAM. CBAM is typically performed in two stages, first to give a rough estimate of cost and benefit and the second one to operate on a smaller set of scenarios and architectural decisions in order to examine them in greater detail.

Dolan has created a method that is specialized in assessing information-system families [21, 34, 20]. This method, called Family Architecture Assessment Method (FAAM), focuses on two related quality aspects: interoperability and extensibility of multiple related systems. Family stakeholders present the business context of the software architecture domain, to provide and rank the system requirements and decide upon the assessment results. This method is very different from all other methods that have been discussed, because it assesses multiple software systems at the same time.
Bengtsson and Bosch have developed a method that analyzes maintainability of a software system by looking at the impact of scenarios at the software architecture level [7, 19]. The total maintenance effort is predicted by summing up the size of the impact of the scenarios multiplied by their probability. In comparison to modifiability assessment methods (e.g. ALMA), maintainability focuses more on repairing bugs, whereas modifiability indicates addition of new functionality to a software architecture.

These methods can be used to examine three different interesting aspects of a software architecture. These aspects are however not applicable to this project. Only the approach of these methods can be used to improve, for example, our usability assessment. The methods in itself are focussing on parts of an architecture in which we are (currently) not interested.
Chapter 8  

Concluding Remarks

This thesis describes the project of adapting GridAssist with a framework for interoperability with grid middleware. An introduction to grid computing, grid middleware and software architecture assessment methods has been given. In addition, also the functions that GridAssist provides to the grid community have been described. This project has adapted GridAssist and delivered a prototype that uses multiple different wrappers for communication with multiple grid middleware environments. The design process was followed by an assessment of the quality attributes and the adaptations to the design. For prototyping purposes wrappers for GT2 and GT4 have been implemented (see appendix C), which were tested on a testbed with multiple middleware environments. And finally, a user walkthrough was performed to gather information of real-life user experience with the prototype. My focus was especially on the influence of the architectural adaptations on GridAssist’s usability.

The remainder of this chapter will give a discussion on the design, on grid computing and on the architecture assessment. Furthermore, some recommendations for future work are given.

8.1 Discussion

This section discusses the work that has been done during this thesis. Also some important developments and remarks are brought under attention.

8.1.1 The Global Grid

One interoperable grid is still far away, because there are too many un-interoperable technologies. Because the GGF is developing standards for all facets of grids, the
technologies are coming closer and closer together. For example, the gridFTP transfer protocol is now used by Globus Toolkit, gLite and also by UNICORE. This makes it easier and easier to combine the grid middleware environments. The current way of grid developments looks very promising, and maybe within time a true global grid will be a fact.

8.1.2 Architecture Assessment

Software architecture assessment is a field wherein quite a lot of research has been done. This can be seen by the high number of methods and the amount of academic publications. Despite this high availability of methods, the software architecture assessment stays a custom-fit operation. For each specific software architecture a suitable assessment methods need to be chosen, combined from existing methods or created from scratch. This can be seen as a drawback, because it can take a lot of time to tailor a suitable assessment setup. On the other hand, it gives the freedom to assess exactly those aspects of the architecture under investigation. When an assessment setup has been chosen well, the results should provide the software architect with the answers he or she was looking for. The subjects of these answers can range from quality attribute conformance to degradation of the architecture’s quality.

Usability is the most important quality attribute of GridAssist. Therefore, the software architecture assessment had a strong focus on this attribute. However, usability is one of the quality attributes that is difficult to grasp during the architectural design phase. The user interface will be fully established during detailed design, so not all details are available during the assessment. Furthermore, usability is dependent on the user that is working with the application. What is usable for user A, might be too complex for user B. This relation with users is hard to grasp during architectural design, because there is not yet a prototype to present to users.

The software architecture assessment has been applied on quite a small project. Small software systems usually don’t have complex relations between its architecture and the quality of the system. Therefore, I think an assessment is more appropriate in larger software projects where the quality of the system is not as predictable as it is for small applications, such as GridAssist.

For an objective assessment it is necessary that the assessor is someone who is not related to the design process. Only then will the assessor be able to focus on the software qualities without being distracted by its own knowledge of the design process. During this assessment, the role of architect was fulfilled by the same per-
son who performed the assessment. This may have caused inconsistencies in the software assessment, as this project was unfortunately only a one-man project. In larger projects when more people are available, the tasks of architect and assessor should be separated.

Currently, there is no large user community available for GridAssist. More users will normally give more feedback, which provides more accurate information on the quality attributes of GridAssist. A large user community is necessary for an accurate assessment and gives results through extensive interviews, walkthroughs and field observations [32]. More users will also influence the effectiveness of a user walkthrough, which is especially useful for verifying usability predictions based on the software architecture. Therefore it was a disadvantage that we lacked a large set of users. More participating users would certainly have increased the accuracy of the usability investigation.

Concluding, I think a software architecture assessment is a versatile method that, because it is customized for each setup, allows all kinds of assessments of a software architecture. For GridAssist, the assessment has made clear that efficiency and reliability aspects of usability are conform the requirements. Furthermore, it was shown that the design adaptations influence screen complexity in a large part of the scenarios, but with only a small complexity increase. Furthermore, I am convinced that all other aspects of an architecture can be examined, as long as an appropriate assessment setup can be constructed.

8.2 Future Work

The improvement and implementation of GridAssist remains an ongoing process. There are still a lot of improvements possible and the developments in grid technology demand an ongoing adaptation. The GGF keeps working on standardization and as they get more successful, more grid technology will get standardized which would require adaptation by GridAssist.

For now, wrappers for the Globus Toolkit version 2.4 and 4.0 have been provided. These two grid environments cover a large part of current grid nodes and will also cover the majority of the future of grids. But additional wrappers can be implemented for grids that are worth the effort. A research was performed to indicate the applicability and possibilities with the gLite middleware, but also UNICORE, the 'BigTop' Microsoft Grid [24] and the GridBus [14] project can be used for integration into GridAssist. This way GridAssist can become a universal viewing window for
all grid initiatives. When all the separate grid initiatives unite into one global grid, GridAssist can become its graphical user interface. In that situation GridAssist could be for the grid what Internet Explorer is for the World Wide Web. However, before we are that far, GridAssist should be extended to cope with additional grid functionality such as dynamic scheduling, resource monitoring, file replication, etc.

Furthermore, the results of the scenario-based assessment of usability have indicated that there are some points of improvement possible. The support of the software architecture for usability sub-attributes satisfaction and learnability is not optimal and could be improved in order to fully meet the desired usability level. This can be done by adding usability patterns and properties to the software architecture. Learnability should be improved for scenarios 'add certificate' and 'change certificate', for example by adding patterns like 'multi-channeling' or 'multiple views'. Satisfaction needs to be improved for scenario 'view resource' by adding patterns like 'cancel', 'undo' or 'user profile'. In addition, there are a few scenarios that will benefit from a decrease of the complexity of the elements and windows, which can be realized by hiding expert-features behind a button. This will decrease the scenario’s complexity for non-expert usage.

Finally, it is an interesting idea to combine the two elements from this thesis; use the software architecture assessment methods to assess the applicability of the grid middleware environments. This will no longer be an architecture assessment, but rather a grid middleware assessment. The same techniques might be applicable and with minor adaptations the same methods could be used. This will allow a solid choice for grid middleware that is to be added to GridAssist. Which guarantees that future developments of GridAssist will be done with the most appropriate grid middleware. A thorough assessment will certainly improve GridAssist’s adoption of grid middleware and increase its competitive position.
Bibliography


Appendix A

Acronyms

The literature of grid computing and architecture assessment is flooded with acronyms. For consistency reasons, most of the acronyms and abbreviations are also used in this thesis. This appendix gives an explanation of the acronyms and abbreviations that are used throughout this thesis.

ADD Architectural Design Document
AJO Abstract Job Objects
ALMA Architecture Level Modifiability Analysis
ALPSM Architecture Level Prediction of Software Maintenance
API Application Programmers Interface
ASP Application Service Provider
ASR Abstract Status Request
ATAM Architectural Trade-Off Analysis Method
CBAM Cost Benefit Analysis Method
CE Computing Element
CoG Commodity of Grid toolkit
CPU Central Processing Unit
DAS-2 Second Distributed ASCI Supercomputer
DDD Detailed Design Document
ECSS European Cooperation for Space Standardization
EDG European Data Grid
EGEE Enabling Grids for European E-Science
ESA European Space Agency
FAAM Family Architecture Assessment Method
FTP File Transfer Protocol
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GASS</td>
<td>Global Access to Secondary Storage</td>
</tr>
<tr>
<td>GAT</td>
<td>Grid Application Toolkit</td>
</tr>
<tr>
<td>GGF</td>
<td>Global Grid Forum</td>
</tr>
<tr>
<td>GQM</td>
<td>Goal Question Metric</td>
</tr>
<tr>
<td>GRAM</td>
<td>Grid Resource Allocation Manager</td>
</tr>
<tr>
<td>GSI</td>
<td>Grid Security Infrastructure</td>
</tr>
<tr>
<td>GT1</td>
<td>Globus Toolkit version 1</td>
</tr>
<tr>
<td>GT2</td>
<td>Globus Toolkit version 2</td>
</tr>
<tr>
<td>GT3</td>
<td>Globus Toolkit version 3</td>
</tr>
<tr>
<td>GT4</td>
<td>Globus Toolkit version 4</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IO</td>
<td>Input-Output</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JDL</td>
<td>Job Description Language</td>
</tr>
<tr>
<td>JMC</td>
<td>Job Monitoring Controller</td>
</tr>
<tr>
<td>JPA</td>
<td>Job Preparation Agent</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>KNMI</td>
<td>Koninklijk Nederlands Meteorologisch Instituut</td>
</tr>
<tr>
<td>LCG</td>
<td>Large Hadron Collider Computing Grid</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LHC</td>
<td>Large Hadron Collider</td>
</tr>
<tr>
<td>LTS</td>
<td>Local Transfer Service</td>
</tr>
<tr>
<td>MDS</td>
<td>Monitoring and Discovery Service</td>
</tr>
<tr>
<td>NJS</td>
<td>Network Job Supervisor</td>
</tr>
<tr>
<td>OGSA-DAI</td>
<td>Open Grid Service Architecture-Data Access and Integration</td>
</tr>
<tr>
<td>OGSA</td>
<td>Open Grid Service Architecture</td>
</tr>
<tr>
<td>OGSI</td>
<td>Open Grid Service Infrastructure</td>
</tr>
<tr>
<td>OMI</td>
<td>Ozone Monitoring Instrument</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PSS-05</td>
<td>Procedures Standards and Specifications for Software Engineering</td>
</tr>
<tr>
<td>R-GMA</td>
<td>Relational Grid Monitoring Architecture</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>RSL</td>
<td>Resource Specification Language</td>
</tr>
</tbody>
</table>
SAAM  Software Architecture Assessment Method
SACU  Scenario-based Architecture Comparison of Usability
SAEM  Software Architecture Evaluation Model
SALUTA  Scenario Based Architectural Level Usability Analysis
SBAR  Scenario-Based Architecture Re-engineering
SOA  Service Oriented Architecture
SOAP  Simple Object Access Protocol
SQUASH  Systematic Quantitative Analysis of Scenarios’ Heuristics
SRD  Software Requirements Document
SRM  Storage Resource Manager
SSL  Secure Socket Layer
TNO  Netherlands Organisation for Applied Scientific Research
TSI  Target System Interfaces
UNICORE  Uniform Interface to Computing Resources
UPL  UNICORE Protocol Layer
URD  User Requirements Document
USite  UNICORE Site
UUDB  UNICORE User Database
VO  Virtual Organization
VOMS  Virtual Organization Management System
VSite  Virtual Site
WMS  Workload Management Service
WN  Worker Node
WS  Web Service
WSDL  Web Service Description Language
WSDM  Web Service Distributed Management
WSRF  Web Service Resource Framework
XML  eXtensible Markup Language
Appendix B

Requirements on Design

This appendix lists all the user and software requirements for the design phase of the adaptations to GridAssist. The functional user requirements describe what the user expects the system to do and are listed in table B.1. The quality user requirements (i.e. how the system is to do its functions) are described in table B.2. From these user requirements the software requirements were derived. These requirements state the necessities of the system, for functionality in table B.3 and in respect to quality of the service in table B.4.

Functional User Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Subject</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UR01</td>
<td>Execution</td>
<td>The GridAssist system SHALL be able to run GridAssist applications on computers that have Globus Toolkit 2.x installed as the computing nodes and/or storage nodes without modifications to the middleware.</td>
</tr>
<tr>
<td>UR02</td>
<td>Execution</td>
<td>The GridAssist system SHALL be able to run GridAssist applications on computers that have Globus Toolkit 4.0 installed as the computing nodes and/or storage nodes without modifications to the middleware.</td>
</tr>
<tr>
<td>UR03</td>
<td>Execution</td>
<td>The GridAssist system SHOULD be able to run GridAssist applications on computers that have EGEE (gLite) installed as the computing nodes and/or storage nodes without modifications to the middleware.</td>
</tr>
<tr>
<td>UR04</td>
<td>Execution</td>
<td>The GridAssist system SHOULD be able to run GridAssist applications on computers that have UNICORE installed as the computing nodes and/or storage nodes without modifications to the middleware.</td>
</tr>
<tr>
<td>UR05</td>
<td>Workflow</td>
<td>The user, developer and administrator SHALL be able to construct, submit and monitor a workflow that is to run within a single Grid Middleware Environment.</td>
</tr>
<tr>
<td>UR06</td>
<td>Workflow</td>
<td>For intra-environmental workflow, the user, developer and administrator SHALL be able to choose the Grid Middleware environment on which the workflow should be executed.</td>
</tr>
</tbody>
</table>
Appendix B. Requirements on Design

| UR07 | Workflow | The user, developer and administrator SHOULD be able to construct, submit and monitor a workflow that uses resources from different Grid Middleware environments. The guest, user, developer and administrator SHALL be able to view the Grid Middleware Environment type of the resource. |
| UR08 | Middleware type | The developer and administrator SHALL be able to change the Grid Middleware Environment type of the resource to one of the predefined Grid Middleware Environment types. The administrator SHOULD be able to add a Grid Middleware Environment type. |
| UR09 | Middleware type | The developer and administrator SHALL be able to change the Grid Middleware Environment type of the resource to one of the predefined Grid Middleware Environment types. The administrator SHOULD be able to add a Grid Middleware Environment type. |
| UR10 | Middleware type | The developer and administrator SHALL be able to assign a Grid Middleware Environment type to a resource upon entry in the registry. The user SHALL be able to define a file transfer in his workflow between two grid data resources that have the same middleware implemented. |
| UR11 | Middleware type | The developer and administrator SHALL be able to assign a Grid Middleware Environment type to a resource upon entry in the registry. The user SHALL be able to define a file transfer in his workflow between two grid data resources that have the same middleware implemented. |
| UR12 | Middleware type | The developer and administrator SHALL be able to assign a Grid Middleware Environment type to a resource upon entry in the registry. The user SHALL be able to define a file transfer in his workflow between two grid data resources that have the same middleware implemented. |
| UR13 | Transfer | The user SHALL be able to transfer files from a data storage resource to the computer of the end user, independent of the grid middleware type. The user SHOULD be able to define a file transfer in his workflow between two grid data resources that have different middleware types implemented. |
| UR14 | Transfer | The user SHALL be able to define a file transfer in his workflow between two grid data resources that have different middleware types implemented. |

Table B.1 — Functional User Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Subject</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UR20</td>
<td>Performance</td>
<td>The current level of performance (controller processing time) SHALL be maintained within certain parameters. In other words, the system’s performance SHALL not degrade more than 5 percent.</td>
</tr>
<tr>
<td>UR21</td>
<td>Performance</td>
<td>The overhead measure is the relationship between the job submission time and the job execution time. Overhead SHALL not increase more than 5 percent.</td>
</tr>
<tr>
<td>UR22</td>
<td>Portability</td>
<td>Portability of GridAssist workflow tool and controller SHALL not be affected by adaptations to the current implementation. The tool SHALL still work on computers running Linux, UNIX, Macintosh or Windows. It SHALL be possible to use the tools of GridAssist from a simple desktop computer without having special middleware installed on this computer.</td>
</tr>
<tr>
<td>UR23</td>
<td>Portability</td>
<td>Portability of GridAssist workflow tool and controller SHALL not be affected by adaptations to the current implementation. The tool SHALL still work on computers running Linux, UNIX, Macintosh or Windows. It SHALL be possible to use the tools of GridAssist from a simple desktop computer without having special middleware installed on this computer.</td>
</tr>
<tr>
<td>UR24</td>
<td>Security</td>
<td>The current level of security (authentication with certificates) should be maintained. In other words, the system SHALL not get less secure. Users need to be authenticated when using GridAssist and authorized to perform the requested actions. The adaptations to GridAssist SHALL adapt the user interface by affecting only 2 graphical window changes.</td>
</tr>
<tr>
<td>UR25</td>
<td>Usability</td>
<td>The adaptations to GridAssist SHALL adapt the user interface by adding 6 graphical window changes. The user SHOULD be able to use GridAssist without possessing technical knowledge of the supported middleware types, their possibilities, requirements and limitations.</td>
</tr>
<tr>
<td>UR26</td>
<td>Usability</td>
<td>The adaptations to GridAssist SHALL adapt the user interface by adding 6 graphical window changes. The user SHOULD be able to use GridAssist without possessing technical knowledge of the supported middleware types, their possibilities, requirements and limitations.</td>
</tr>
<tr>
<td>UR27</td>
<td>Usability</td>
<td>The adaptations to GridAssist SHALL adapt the user interface by adding 6 graphical window changes. The user SHOULD be able to use GridAssist without possessing technical knowledge of the supported middleware types, their possibilities, requirements and limitations.</td>
</tr>
</tbody>
</table>

Table B.2 — Quality User Requirements
### Functional Software Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Subject</th>
<th>Description</th>
<th>User Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR01</td>
<td>Middleware</td>
<td>The system SHALL contain a framework for plug-in wrappers, each supporting a Grid middleware environment.</td>
<td>UR05</td>
</tr>
<tr>
<td>SR02</td>
<td>Middleware</td>
<td>The middleware framework SHALL need at least one plug-in wrapper for operation. Wrappers are presented as plug-ins and can be plugged in at run time by the administrator.</td>
<td>UR06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR04</td>
</tr>
<tr>
<td>SR03</td>
<td>Middleware</td>
<td>A Globus Toolkit 2.x wrapper SHALL be implemented for operation with the Globus Toolkit version 2 grids.</td>
<td>UR01</td>
</tr>
<tr>
<td>SR04</td>
<td>Middleware</td>
<td>A Globus Toolkit 4 wrapper SHALL be implemented for operation with the Globus Toolkit version 4 grids.</td>
<td>UR02</td>
</tr>
<tr>
<td>SR05</td>
<td>Middleware</td>
<td>A UNICORE wrapper SHOULD be implemented for operation with the UNICORE grid.</td>
<td>UR03</td>
</tr>
<tr>
<td>SR06</td>
<td>Middleware</td>
<td>An EGEE (gLite) wrapper SHOULD be implemented for operation with the EGEE (gLite) grid.</td>
<td>UR04</td>
</tr>
<tr>
<td>SR07</td>
<td>Middleware</td>
<td>The middleware framework SHALL act as a uniform interface to Grid Middleware for the other GridAssist classes.</td>
<td>UR01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR04</td>
</tr>
<tr>
<td>SR09</td>
<td>Middleware</td>
<td>A wrapper SHALL map all functions of GridAssist onto calls for one Grid Middleware implementation. In other words, a wrapper SHALL implement at least all necessary functionality (workflow submission, credential creation, file transferring and job monitoring).</td>
<td>UR01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR04</td>
</tr>
<tr>
<td>SR10</td>
<td>Resource</td>
<td>The system SHALL allow all users to view the type of each resource.</td>
<td>UR08</td>
</tr>
<tr>
<td>SR11</td>
<td>Resource</td>
<td>The system SHALL allow the administrator to change the type of each resource.</td>
<td>UR09</td>
</tr>
<tr>
<td>SR12</td>
<td>Resource</td>
<td>The system SHALL allow the administrator to add a resource and thereby specifying the resource type.</td>
<td>UR10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR11</td>
</tr>
<tr>
<td>SR13</td>
<td>Workflow</td>
<td>The system SHALL allow users, developers and administrators to execute workflows that use only one grid middleware (so also one wrapper).</td>
<td>UR05</td>
</tr>
<tr>
<td>SR14</td>
<td>Workflow</td>
<td>The system SHOULD allow users, developers and administrators to execute workflows that use more than one grid middleware solution (so also more than one wrapper). The controller will divide a workflow over multiple wrappers and will collect multiple results when the job is finished.</td>
<td>UR06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR27</td>
</tr>
<tr>
<td>SR15</td>
<td>Transfer</td>
<td>The system SHALL allow the user to transfer files in his workflow, between two data resources that have the same middleware implemented.</td>
<td>UR12</td>
</tr>
<tr>
<td>SR16</td>
<td>Transfer</td>
<td>The system SHOULD allow the user to transfer files in his workflow, between two data resources that have different middleware implemented.</td>
<td>UR13</td>
</tr>
</tbody>
</table>
Appendix B. Requirements on Design

Table B.3 — Functional Software Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Subject</th>
<th>Description</th>
<th>User Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR17</td>
<td>Transfer</td>
<td>The system SHALL allow the user to transfer files between a data resource and the end-user computer, independent of the middleware environment that the data resource has installed. Any legacy design elements, which are unpractical in current perspective of interoperability, SHALL be re-designed to meet current design objectives and requirements. However, this should be done with care, because it is not a project goal to redesign GridAssist.</td>
<td>UR14</td>
</tr>
<tr>
<td>SR18</td>
<td>Re-design</td>
<td>Any legacy design elements, which are unpractical in current perspective of interoperability, SHALL be re-designed to meet current design objectives and requirements. However, this should be done with care, because it is not a project goal to redesign GridAssist.</td>
<td>-</td>
</tr>
</tbody>
</table>

Quality Software Requirements

<table>
<thead>
<tr>
<th>Req</th>
<th>Subject</th>
<th>Description</th>
<th>User Req</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR20</td>
<td>Performance</td>
<td>The overhead measure is the relationship between the job submission time and the job execution time. Overhead SHALL not increase more than 5 percent. So the overhead SHALL be lower than 0.1 seconds per job running second.</td>
<td>UR20, UR21</td>
</tr>
<tr>
<td>SR21</td>
<td>Performance</td>
<td>The Controller (including the middleware framework) of GridAssist SHALL be able to operate on a continuous basis, 24 hours per day, if required.</td>
<td>UR20, UR21</td>
</tr>
<tr>
<td>SR30</td>
<td>Interface</td>
<td>The middleware framework SHALL communicate with Grid Middleware that is Globus based (Globus 2.x and Globus 4.0) using the CoG kit.</td>
<td>-</td>
</tr>
<tr>
<td>SR31</td>
<td>Interface</td>
<td>All communication to one Grid Middleware environment SHALL go through one specific wrapper. This wrapper is specifically designed for its target Grid Middleware environment.</td>
<td>UR01, UR02, UR03, UR04</td>
</tr>
<tr>
<td>SR32</td>
<td>Usability</td>
<td>The user interface component of GridAssist SHALL undergo only minimal changes. In other words, only a small amount (~20%) of user interface elements (windows, input fields, etc.) SHALL be affected by the changes.</td>
<td>UR25, UR26</td>
</tr>
<tr>
<td>SR40</td>
<td>Operational</td>
<td>Upon workflow creation, the user can indicate on which resource to run a workflow job, thereby specifying on which middleware type the job SHALL run.</td>
<td>UR05, UR06, UR25</td>
</tr>
<tr>
<td>SR41</td>
<td>Operational</td>
<td>Upon workflow submission, the workflow tool SHALL be able to run the workflow on any compatible available resource disregarding the Grid Middleware Environment type.</td>
<td>UR06, UR26, UR27, UR08, UR09, UR25</td>
</tr>
<tr>
<td>SR42</td>
<td>Operational</td>
<td>The window that shows resource information SHALL also show the middleware type of the resource.</td>
<td>UR09, UR25</td>
</tr>
<tr>
<td>SR43</td>
<td>Operational</td>
<td>There SHALL be an extra window that allows the administrator to view and edit the middleware types and additional wrappers.</td>
<td>UR09, UR25</td>
</tr>
<tr>
<td>SR050</td>
<td>Resource</td>
<td>The Controller (including the middleware framework) SHALL be able to run on a computer that has an Internet connection, allows HTTP traffic to port 80 and port 8080, allows data traffic to ports used by the grid middleware solutions (Globus 2, Globus 4, EGEE or UNICORE), has at least 256 Mb internal memory, has at least 512 Mb free disk space. The client (workflow tool) SHALL be able to run on a computer that has an Internet connection, allows HTTP traffic to port 80 and port 8080, has at least 128 Mb internal memory, has at least 128 Mb free disk space.</td>
<td></td>
</tr>
<tr>
<td>SR051</td>
<td>Resource</td>
<td>Computers that that can be used as Grid nodes SHALL: have an Internet connection, allow data traffic via the ports used by the Grid middleware (Globus 2, Globus 4, EGEE or UNICORE), have at least 128 Mb internal memory, have at least 512 Mb free disk space, have a Grid Middleware solution (Globus 2, Globus 4, EGEE or UNICORE) installed, provide accounts and proper certificates for the users.</td>
<td></td>
</tr>
<tr>
<td>SR052</td>
<td>Resource</td>
<td>The system SHALL support simulated running of a job. The system does not need any wrapper plug-ins for a simulated run. This verification test is to test the internal workings of workflow submission in GridAssist.</td>
<td></td>
</tr>
<tr>
<td>SR060</td>
<td>Verification</td>
<td>The system SHALL support testing of wrapper plug-ins. In this test new wrappers are detected and their input-output relation and performance is measured. The GridAssist Interoperability system SHALL be documented as described in the &quot;GridAssist Interoperability Project Management Plan&quot;. The system SHALL be able to handle Globus user certificates.</td>
<td></td>
</tr>
<tr>
<td>SR061</td>
<td>Verification</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SR080</td>
<td>Documentation</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SR090</td>
<td>Security</td>
<td>The system SHALL be able to handle Globus user certificates.</td>
<td></td>
</tr>
<tr>
<td>SR091</td>
<td>Security</td>
<td>The system SHOULD be able to handle EGEE user certificates.</td>
<td></td>
</tr>
<tr>
<td>SR092</td>
<td>Security</td>
<td>The system SHOULD be able to handle UNICORE user certificates.</td>
<td></td>
</tr>
<tr>
<td>SR093</td>
<td>Security</td>
<td>The system SHALL be able to accept certificates issued by the DutchGrid CA.</td>
<td></td>
</tr>
<tr>
<td>SR094</td>
<td>Security</td>
<td>The system SHALL posses (get from the user) at least one certificate for each middleware environment the user likes to access.</td>
<td></td>
</tr>
<tr>
<td>SR095</td>
<td>Security</td>
<td>The system SHOULD store certificates and pass-phrases, so that certificates can be generated by the system for user authentication on the grid.</td>
<td></td>
</tr>
<tr>
<td>SR100</td>
<td>Portability</td>
<td>The system SHALL be implemented in Java 1.4 to ensure portability between Linux, UNIX and windows platforms. The system SHALL run on Linux, UNIX and Windows without necessary modifications.</td>
<td></td>
</tr>
</tbody>
</table>
The system **SHALL** re-schedule a job to another grid node with the same Grid middleware implementation when due to a failure of hardware and/or software a job on a selected node cannot be completed.

The system **SHOULD** re-schedule a job to another Grid middleware environment when (after several retries) the job keeps failing on the intended Grid middleware environment.

The system **SHALL** allow extension of the middleware framework with additional middleware wrappers without any required changes to the internal workings of the system.

The system **SHOULD** allow extension of the middleware types, from the user interface (through plug-ins).

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Quality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR120</td>
<td>Reliability</td>
<td>The system SHALL re-schedule a job to another grid node with the same Grid middleware implementation when due to a failure of hardware and/or software a job on a selected node cannot be completed.</td>
</tr>
<tr>
<td>SR121</td>
<td>Reliability</td>
<td>The system SHOULD re-schedule a job to another Grid middleware environment when (after several retries) the job keeps failing on the intended Grid middleware environment.</td>
</tr>
<tr>
<td>SR130</td>
<td>Modifiability</td>
<td>The system SHALL allow extension of the middleware framework with additional middleware wrappers without any required changes to the internal workings of the system.</td>
</tr>
<tr>
<td>SR131</td>
<td>Modifiability</td>
<td>The system SHOULD allow extension of the middleware types, from the user interface (through plug-ins).</td>
</tr>
</tbody>
</table>

Table B.4 — Quality Software Requirements
This appendix gives an impression of the source code of a wrapper for a grid middleware environment. As an example the wrapper for GT4 has been given. This appendix is also a suitable start for people investigating GT4 communication. For better visualization, the list of imports statements has been omitted, but they include most of the libraries from the GT4 installation with some additions for credentials and file transfers.

```java
public class Globus4Wrapper implements Wrapper {

    private GlobusCredential cred;
    private GSSCredential gsscred;
    private static final int WAIT_BETWEEN_POLL = 1000;
    private EndpointReferenceType credentialEndpoint;
    private String credentialEndpointResource;
    private ManagedJobFactoryPortType factoryPort;
    private ClientSecurityDescriptor secDesc;
    private ManagedJobPortType jobport;
    private boolean keeprunning;

    /**
     * Constructor
     */
    public Globus4Wrapper() {
        // setup SSL
        Util.registerTransport();
        cred = null;
        gsscred = null;
        credentialEndpoint = null;
        credentialEndpointResource = "";
    }

```
keeprunning = true;
}

/**
 * Returns the name of this wrapper
 * @return the name of this wrapper
 */
public String getName() {
    return "Globus 4.0.0 Wrapper";
}

/**
 * Create the security description object, which is used by the SSL communication.
 */
private void initSecurityDescription() {
    secDesc = new ClientSecurityDescriptor();
    secDesc.setGSITransport(GSIConstants.MODE_SSL);
    secDesc.setGSSCredential(gsscred);
    secDesc.setDelegation(new HostAuthorization());
    secDesc.setDelegation(org.globus.axis.gsi.GSIConstants.GSI_MODE_FULL_DELEG);
}

/**
 * Sets the GSSCredential for the wrapper
 * @param gsscred the credential to set
 */
public void setCredential(GSSCredential gsscred) {
    this.gsscred = gsscred;
    // initialise security description with the above credential
    initSecurityDescription();
}

/**
 * Sets the GSSCredential for this wrapper
 * @param c Credential to set (CredentialMyProxy, CredentialGlobusProxy,
 * CredentialX509Certificate)
 */
public void setCredential(Credential c) throws Exception {
    if (c.getClass().getName().endsWith("CredentialMyProxy")) {
        CredentialMyProxy proxy = (CredentialMyProxy)c;
        if (proxy != null) {
            proxy.getCredential();
            this.gsscred = proxy.cred;
        }
    }
    else if (c.getClass().getName().endsWith("CredentialGlobusProxy")) {
        CredentialGlobusProxy proxy = (CredentialGlobusProxy)c;
        if (proxy != null) {
            this.gsscred = proxy.getCredential();
        }
    }
else if (c.getClass().getName().endsWith("CredentialX509Certificate")) {
    System.out.println("Globus4Wrapper.java : setCredential() : found x509 cert");
    CredentialX509Certificate proxy = (CredentialX509Certificate)c;
    if (proxy != null) {
        //backup InputStreams
        byte[] certbytes = new byte[proxy.certificateFile.available()];
        byte[] keybytes = new byte[proxy.keyFile.available()];
        proxy.certificateFile.read(certbytes);
        proxy.keyFile.read(keybytes);
        proxy.certificateFile = (InputStream)new ByteArrayInputStream(certbytes);
        proxy.keyFile = (InputStream)new ByteArrayInputStream(keybytes);
        InputStream[] f = {new ByteArrayInputStream(certbytes),
                          new ByteArrayInputStream(keybytes)};
        setCredential(f, proxy.passphrase);
    }
} }

/**
* Create local credential.
* @param f InputStreams of the files needed for credential generation.
* @param passphrase The password of the private key file.
*/
public void setCredential(InputStream f[], String passphrase) throws Exception {
    try {
        int validity = 43200; // default 12 hours
        X509Certificate certToSign2 = CertUtil.loadCertificate(f[0]);
        OpenSSLKey key = new BouncyCastleOpenSSLKey(f[1]);
        key.decrypt(passphrase);
        PrivateKey userKey = key.getPrivateKey();
        BouncyCastleCertProcessingFactory factory =
            BouncyCastleCertProcessingFactory.getDefault();
        X509Certificate[] certList = { certToSign2 };
        cred = factory.createCredential(certList, userKey, 512, validity,
                                         GSIContstants.GSI_2_PROXY);
        gsscred = new GlobusGSSCredentialImpl(cred, GSSCredential.INITIATE_AND_ACCEPT);
        // initialise security description with the above credentials
        initSecurityDescription();
    } catch(Exception e) {
        //System.out.println(e.getMessage());
        e.printStackTrace();
        throw new Exception("Cannot create credential");
    }
} }

/**
* Submit the credential to the resource and retrieve a endpoint.
 ```java
private void stageCredential(String resourceUrl) throws Exception {
    EndpointReferenceType delegFactoryEndpoint = new EndpointReferenceType(new Address(resourceUrl + "/wsrf/services/DelegationFactoryService"));
    X509Certificate[] certToSignArray = DelegationUtil.getCertificateChainRP(delegFactoryEndpoint, secDesc);
    X509Certificate certToSign = certToSignArray[0];
    credentialEndpoint = DelegationUtil.delegate(resourceUrl + "/wsrf/services/DelegationFactoryService", cred, certToSign, true, secDesc);
    credentialEndpointResource = resourceUrl;
}

/** 
 * @param resourceUrl the url of the resource on which the job is to run 
 * @param rsl xml representation of the job 
 * @throws Exception When job submission or job execution fails. 
 */
public void run(String resourceUrl, String rsl) throws Exception {
}

/** 
 * Submit job to resource 
 * @param resourceUrl The url of the resource on which the job is to run. 
 * @param executable The path of the executable on the resource. For example /bin/sh. 
 * @param arguments Array with strings which are used as input for the executable. 
 * @param output 
 * @param dir 
 * @param jobType 
 * @param queue 
 * @throws Exception When job submission or job execution fails. 
 */
public void run(String resourceUrl, String jobManager, String executable, String[] arguments, String output, String dir, String jobType, String queue) throws Exception {
    if(!checkCredential()) throw new Exception("No valid credentials");
    String contactString = resourceUrl + "/wsrf/services/ManagedJobFactoryService";
    URL factoryURL = ManagedJobFactoryClientHelper.getServiceURL(contactString).getURL();
    String factoryType = ManagedJobFactoryConstants.FACTORY_TYPE.FORK;
    EndpointReferenceType factoryEndpoint = ManagedJobFactoryClientHelper.getFactoryEndpoint(factoryURL, factoryType);
    factoryPort = ManagedJobFactoryClientHelper.getPort(factoryEndpoint);
    // send credential to resource if not already done
    if(!credentialEndpointResource.equals(resourceUrl)) stageCredential(resourceUrl);

    // set security for portType
    ((javax.xml.rpc.Stub)factoryPort)._setProperty(
        org.globus.wsrf.security.Constants.CLIENT_DESCRIPTOR, secDesc);

    CreateManagedJobInputType jobInput = new CreateManagedJobInputType();
```
Calendar c = Calendar.getInstance();
int minutes = 60;
c.setTimeInMillis(c.getTimeInMillis() + minutes*60000);
jobInput.setInitialTerminationTime(c);

// Setting up the description of the job
JobDescriptionType jobDescrip = new JobDescriptionType();
jobDescrip.setExecutable(executable);
jobDescrip.setStderr("stderr");
if(arguments.length > 0)
    jobDescrip.setArgument(arguments);
if(!output.equals(""))
    jobDescrip.setStdout(output);
else
    jobDescrip.setStdout("stdout");
if(!dir.equals(""))
    jobDescrip.setDirectory(dir);
if(!queue.equals("") \&\& !queue.equals("null"))
    jobDescrip.setQueue(queue);
if(!jobType.equals("")) {
    
}
if(!jobManager.equals("")) {
    
}
jobDescrip.setJobCredentialEndpoint(credentialEndpoint);
jobInput.setJob(jobDescrip);
CreateManagedJobOutputType createResponse = factoryPort.createManagedJob(jobInput);
EndpointReferenceType jobEndpoint = createResponse.getManagedJobEndpoint();
jobport = ManagedJobClientHelper.getPort(jobEndpoint);
// set security for job port
((javax.xml.rpc.Stub)jobport)._setProperty(
    org.globus.warf.security.Constants.CLIENT_DESCRIPTOR, secDesc);
keeprunning = true;
boolean faulted = false;
while(keeprunning) {
    try {
        GetResourcePropertyResponse response =
            jobport.getResourceProperty(ManagedJobConstants.RP_STATE);
        Thread.sleep(WAIT_BETWEEN_POLL);
        if(response.get_any()[0].getValue().equals(StateEnumeration._Done)) {
            keeprunning = false;
        }
        else if(response.get_any()[0].getValue().equals(StateEnumeration._Failed)) {
            keeprunning = false;
            faulted = true;
        }
    }
    catch(Exception e) {
        System.out.println(e.getMessage());
        keeprunning = false;
faulted = true;
    throw new Exception("Cannot retrieve job state");
}
}
if(faulted) throw new Exception("Gram Job Failed");

// destroy the job resource
jobport.destroy(new Destroy());
}

/**
 * Stops the current job
 */
public void stopRunning() throws Exception {
    keeprunning = false;
    if(jobport != null){
        // destroy the job resource
        jobport.destroy(new Destroy());
    }
    else throw new Exception("Job does not exists");
}

/**
 * Copy file via gridftp to or from the grid node
 * @param src The url of the source file. For example: gsiftp://venus:2811/stdout
 * @param dest The url of the destination file. For example file:///stdout_copied
 */
public void copy(String src, String dest) throws Exception {
    UrlCopy urlCopy = new UrlCopy();
    urlCopy.setCredentials(gsscred);
    urlCopy.setSourceUrl(new GlobusURL(src));
    urlCopy.setDestinationUrl(new GlobusURL(dest));
    urlCopy.setUseThirdPartyCopy(true);
    urlCopy.copy();
}

/**
 * Copy file via gridftp from source to destination
 * @param srcHost host name of the source file
 * @param srcFile name of the source file
 * @param dstHost host name of the destination file
 * @param dstFile name of the destination file
 */
public void copy(String srcHost, String srcFile, String dstHost,
    String dstFile) throws Exception {
    String srcFile2 = srcFile;
    String srcFile3 = "gsiftp://" + srcHost + "/" + srcFile2;
    String dstFile2 = dstFile;
    String dstFile3 = "gsiftp://" + dstHost + "/" + dstFile2;
    copy(srcFile3, dstFile3);
public String retrieve(String srcURL) throws Exception {
    File temp = File.createTempFile("temp", ".dat");
    copy(srcURL, "file://~/"+temp.getName());
    FileInputStream fis = new FileInputStream(temp.getName());
    int x= fis.available();
    byte b[]= new byte[x];
    fis.read(b);
    temp.delete();
    return new String(b);
}

public boolean checkCredential() {
    if(cred != null && gsscred != null) {
        try {
            return (cred.getTimeLeft() > 0) && (gsscred.getRemainingLifetime() > 0);
        }
        catch(Exception e) {
            return false;
        }
    } else return false;
}
}
This appendix lists the walkthrough setup that has been used, including some comments made during one of the walkthrough sessions. The first part of the walkthrough was performed on the original GridAssist, and the second part (see next page) on the adapted GridAssist. Experience with GridAssist was scored on a one-to-ten scale, for a fair walkthrough only novice users were chosen.

### Walkthrough GridAssist Interoperability

<table>
<thead>
<tr>
<th>General Information</th>
<th>Experience with GridAssist (1-10):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>Leeuwarden Vijf Winkel</td>
</tr>
<tr>
<td>Date:</td>
<td>11- Augustus 2005</td>
</tr>
<tr>
<td>Function:</td>
<td>Software Eng</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original GridAssist</th>
<th>Comment:</th>
<th>Time:</th>
</tr>
</thead>
</table>

Create a Globus proxy certificate. Certificate file and key files are located in `certs`. And pass phrase is `[redacted]`.

In the registry tab, click on `resources`. Right click on resources to create a resource with the following info:

- Name: `Walkthrough Resource`
- CE Address: `aarde.dutchspace.nl`
- SE Address: `aardedutchspace.nl`
- Job Manager: `fork`
- Job Queue: `null`
- Number of jobs: `1`

Using `Click Apply`.

| Walkthrough Interoperability | Aanmaken in tree werkt intuïtief. Werk t goed ?
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terug koppeling op &quot;Apply&quot; knop</td>
</tr>
</tbody>
</table>

In registry, create a service with the following information:

- Name: `walkthrough`
- Type: `Application`
- Timeout: `0`

Using `Click Apply`.

<table>
<thead>
<tr>
<th>Walkthrough Interoperability</th>
<th>Hier is wel vrije koffers.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Save registry&quot; mogelijk te vinden</td>
</tr>
</tbody>
</table>

Deploy this service with on resource `Walkthrough Resource` and path `/home/kb76093/sleepers`

Using `Click Apply`.

<table>
<thead>
<tr>
<th>Walkthrough Interoperability</th>
<th>Niet uit te zien bij omschrijven.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Save registry&quot; mogelijk te vinden</td>
</tr>
</tbody>
</table>

Save registry and reload it.

In workflow tab, create a simple workflow of only the `walkthrough` service.

<table>
<thead>
<tr>
<th>Walkthrough Interoperability</th>
<th>Niet uit te zien bij omschrijven.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Submit knoop laat je te vinden</td>
</tr>
</tbody>
</table>

Submit the workflow with job description: `walkthrough job`.

<table>
<thead>
<tr>
<th>Walkthrough Interoperability</th>
<th>Terug koppeling hier is prima.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Terug koppeling hier is prima.</td>
</tr>
</tbody>
</table>

Monitor the job until it completes.
<table>
<thead>
<tr>
<th>Adapted GridAssist</th>
<th>Comment:</th>
<th>Time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a credential of type X509 Certificate. Certificate file and key files are located in D:\certs. And pass phrase is <em>REDACTED</em></td>
<td><strong>Browse button achter invoervelden</strong></td>
<td></td>
</tr>
<tr>
<td>In the registry tab, click on middleware type. Right click to add a new middleware type with the following information:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name: test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wrapper class: Globus2Wrapper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Click Apply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save registry and reload it.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In the registry tab, click on the Walkthrough Resource and change the middleware type from gLite to test.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Click Apply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save registry and reload it.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In workflow tab, create a simple workflow of only the walkthrough service.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit the workflow with job description: walkthrough 2 job.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor the job until it completes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**General Comments**

Perceived difference between versions:

- Interface & philosophy is ongeveer tast. Dat bevoordeelt de leesbaarheid.

Which one did you experience as more usable:

-

Complexity difference:

- Gebruiker wil eigenlijk niets weten van middleware/deployments en resources. Dit is goed verborgen in Registry tab.
You do ill if you praise, but worse if you censure, what you do not understand.
- Leonardo da Vinci