Integration of Grid Resources in the RCE Environment

Kenneth Rohde Christiansen[1]
Department of Mathematics and Computing Science
Rijksuniversiteit Groningen
Blauwborgje 3
NL-9747 AC Groningen
The Netherlands

Abstract

SESIS [SESIS] is a conceptual design and simulation system for the early design phases of ship development. The system empowers the engineers to perform complex collaborative simulations between the shipyards and suppliers over the internet.

As part of the SESIS, the Reconfigurable Computing Environment (RCE) platform is developed, which serve as the base system for design and simulation that can be extended by third-party developers by using the RCE Software Development Kit.

The purpose of the thesis is analyzing how and if the RCE platform can take advantage of grid resources. Various options for integrating Grid middleware in the platform will be presented and one option will be chosen: the option to develop a Grid-aware Software Development Kit. The SDK will have to integrate with the RCE Software Development Kit and at least will be designed to work with two grid middleware systems: Globus and UNICORE as this is a wish of the SESIS project.

The requirement of such an SDK will be analysed and the API of the Grid-aware SDK will be designed and a Globus backend will be implemented.

Keywords:

Grid Computing, Distributed Computing, Web Services, Design and Simulation Systems, Software Engineering, Computer Science.

To read this thesis a good level of English and basic knowledge of English computer and software terminology is needed. The thesis contains a terminology glossary at the end that can be used to get acquainted with the domain-specific terminology used within the work domain of this thesis.

RuG Rijksuniversiteit Groningen   DLR Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft

[1] Contact info: Kenneth.Christiansen@dlr.de / Kenneth.Christiansen@gmail.com

Engineering is the art or science of making practical

Samuel C. Florman

Contents

Chapter 1

Introduction

1.1 Forewords

The following thesis represents approximately half a year of work learning about grid technology and middleware as well as the RCE platform (part of SESIS) with the purpose of analyzing how and if the RCE platform could take advantage of grid resources. It also describes the design and implementation of a grid-aware software development kit for use with the RCE platform.

Before the start of the thesis it was already set in stone that it should at least be possible to delegate computational tasks to a grid middleware system, as this has been requested by the customers of the SESIS project. It was also set in stone that the implementation should be done with the Globus grid middleware system, and that it should be possible to write a backend for the competing grid middleware system UNICORE with only minimal changes to what has been implemented during the thesis period.

As new to grid technology, a large part of the thesis period has been used for technology study, often learning about things that turned out to have little relevance to this thesis. Due to this, this thesis also serves as a good introduction to grid technology so that a future maintainer of the developed software will learn what he/she needs to know in no time.

Though that the choices made as a result of technology study might seem more or less obvious, it is important that the study was done in order to know that the right decisions have been made and that the research institute DLR (German Aerospace Center) won't have to deal with maintenance problems for the years to come.

The result of the thesis, in terms of software is a software development kit for taking advantage of grid resources from the RCE platform.

The main features are:

- The possibility to use grid resourced from RCE, using the standard RCE proxy certificate as long as you have been granted right for doing so.
- The possibility to describe and run jobs with a Java-based abstraction API that fits in with the RCE framework. This included handing of file staging (copying input and output files between client and grid host), as well as output/input handling.
- The possibility to transfer files between grid nodes (including client) without actually executing jobs.
- The possibility of resource querying to insure that a host fulfils the requirements of the job.

- The option to implement another backend of the API so that other grid middleware systems can be supported (Other versions of Globus, UNICORE, etc)

What this exactly will say should be clearer when reading the thesis.

The thesis has been structured the following way:

In *chapter 1* I introduce the reader to the SESIS project, which is the project where this thesis makes part of. In order to describe the project better, various case-scenarios are presented.

Knowing the basics about SESIS should make it easier to imagine how grid technology will be used by the customers of SESIS, but as many parts of SESIS has only little relevance to this thesis, it is only touched shortly. More interesting is the RCE platform that SESIS is build on top of, so the chapter also devotes a section to explaining the relation between RCE and SESIS.

In *chapter 2* I start out with a general introduction to grid technology; the abilities, the history, as well as some thoughts about the future prospects. This serves as an introduction to grid technology in general which should help the reader understand enough about grids to read the thesis without limiting the knowledge to just the parts of grid technology that has been used in the implementation of the grid-aware SDK. I hope that this general introduction will save a future maintainer of this developed SDK time getting acquainted with grid technology and might make the maintainer see other uses that I have not thought about.

In the chapter I also look closer at RCE, with a short introduction to the raw design of the platform as well as a section explaining the similarities and differences between RCE and grid middleware systems. This is needed as from the first look at it; the uses of RCE and grid middleware might seem similar, though there are some fundamental differences which fosters the integration of grid support in RCE.

In *chapter 3* I look at where we can apply grid technology to RCE and where it makes the most sense. This serves as part of the overall design decisions; we only want to integrate grid technology where it really makes sense and this needs to be clear before we continue.

The chapter presents various options for add grid support, grouped in 3 groups due to similarities. These options results from a brainstorming meeting between me, Thijs Metsch and Andreas Schreiber. Since the options were only brought up as ideas from the meeting, I look at the options more detailed to find out the advantages and disadvantages of each option. Furthermore I use UML use-cases to better illustrate what the options offers us of features, especially since not all options are mutual exclusive and it is not possible to implement all due to time constraints.

The option to make a grid-aware SDK for RCE seems obvious as the use-cases cover the use-scenarios that the customers of SESIS have requested, though the other options are more technologically interesting.

In *chapter 4,* I look at requirements of such a grid-aware SDK and use more detailed UML use-cases to group the uses together which help me realize that the usages group in 2 groups: system related, data management related and job related.

Based on this knowledge I present the class design that I have made of the grid-aware SDK together with some of the ideas leading to this design. The classes of the public API are shown with the use of UML, together with the classes of the Globus implementation which are shown in a different color. The public API is described as well.

This should help a future developer or maintainer understand the API design, which is needed in order to implement support for different grid middleware systems.

In *chapter 5*, I look closer at how I have implemented the Globus implementation of the public grid-aware API. I perform an analysis to reveal if there are other options than implementing the design by the tedious use of grid services (web services) and I settle on using abstraction layers where possible. A section is also devoted to an analysis of which abstraction layers that are available on the market and their advantages and disadvantages. This has helped me settle on the CoG Toolkit.

The chapter also touches the implementation details in order for a future maintainer to understand that implementation and the choices made. The implementation details are supplied with small, simplified code samples to better illustrate the ideas. The chapter also includes a section about pitfalls and problems encountered while doing the implementation. I hope this will help a future maintainer avoid these pitfalls.

The chapter ends with a section about engineering strategies, which is devoted to introduce a future maintainer to the engineering strategy used when designing and implementing the grid-aware SDK API. It is important to understand this strategy as a future maintainer is required to follow it. These strategies are based on common practice based on my experience within the field of computer science plus additional practices applied here at the German Aerospace Center.

In *chapter 6*, I illustrate how to use the public API of the grid-aware SDK by a few examples. This illustrates the work performed and helps understanding the integration tests described in *chapter 7*.

The last chapters contain evaluation, future work, conclusion and the like.

It should be clear that a lot of effort has been put into technology study and into designing a good design that fits needs of the RCE platform and that does not need to be redesigned in the near future.

1.2 The SESIS Project

In the following section I will introduce the SESIS project that this thesis makes part of. Not everything explained have direct relevance to the thesis, but are still explained shortly in order to give an idea of what the SESIS product is all about. I will also make the difference between the RCE project and the SESIS product clear as this tends to be a source of confusion.

1.2.1 Product Perspective

The goal of the SESIS project is to develop a design-concept and simulation system for building ships. The main objective is to make it easier, quicker and cheaper for the shipyards to develop new ships. This is done by performing complex collaborative simulations between the shipyards and the suppliers in a so-called virtual organization[2], which can be defined as a network of companies, suppliers, customers, or employees, linked by information and communications technologies, with the purpose of delivering a service or product.

The system has to be designed as an integrated and flexible development environment that should be deployable in a distributed computer environment consisting of UNIX, Linux and Windows computers. It should be possible to integrate existing ship development specific software, in order to reuse what already exists.

The project uses the functionality and the experience gained from simulation and design systems for different user domains such as ship, airplane, and vehicle development. It is built upon current future proof software technologies, such as:

- Extendable GUI frameworks
- Wrapping techniques for the integration of legacy code and commercial calculation and simulation applications
- Grid computing

An extendable GUI framework is a collection of graphical elements (so-called widgets) used to build graphical user interfaces that can be customized and extended by combining widgets or extending existing widgets to form new ones.

Wrapping means making it possible to use source code written in another language or complete applications directly from a programming language using native interfaces. There exists standard ways for doing this which are used for the project.

Grid resource will be explained in detain within the next chapter.

The SESIS system is an open system in the way that it is designed to be extended. By the use of clearly defined programming interfaces, third-party developers will be able to develop new *extension methods* as well as integrate already existing methods in the system, so that they can be combined and used together in a so-called workflow chain.

With the preparation of interfaces to commercial simulation tools and other methods and tools such as production planning the functionality needed by the common ship builder should be present in SESIS.

1.2.2 Relation with the Reconfigurable Computing Environment

Before I precede any further I will need to clarify the terminology and point out the difference between RCE and SESIS, as well as how they relate to each other. The work done in this thesis relates to the RCE platform, which is currently only being used by the SESIS project.

[2] Please consult the terminology list at the end of this thesis to get acquainted with the specific terminology used within the work domain of this thesis.

Basically RCE is a platform that offers a distributed system for accessing and managing data, as well as for accessing and using so-called extension methods. These methods are in fact extension components that extent RCE with for instance simulation or calculation methods, which is the case for the SESIS project. RCE is built up around a virtual organization structure which means that a distributed RCE installation, given the appropriate rights, can be used by multiple organizations, enabling them to cooperate by sharing their data and use each others methods.

RCE is in itself not an actual product that will ever be released; however it is a platform that will be used by various products developed at DLR (German Aerospace Center).

SESIS, which is one of these products based on RCE, is a specialized version of RCE supplied with extra standard methods usable when designing ships. In the future other products might be released specialized for design of aircrafts and space shuttles, but for the moment SESIS is the only product being directly worked on and the work of this thesis is thus done as part of the SESIS project.

1.2.3 Case Stories for the SESIS Project

In order to understand exactly what SESIS does I will look at a few of the use-case scenarios supplied by future users of the system. For understanding the following use-cases, basic knowledge of English ship terminology is needed.

Simulation of ship and engine

Scenario supplied by FSG[3] and SAM[4]

A concept engineer wants to simulate the timely progress of the position of a sailing ship as well as the state of the ship engine. The simulation method, which are needed are not available on the local installation of SESIS, will need to be copied from remote installations (servers). The interface and parameters of the methods will need to be described.

The client at SAM simulates an electrical ship engine with the use of the software application Simplorer[5]. The module data for the propeller and the ship resistance are available at an FSG server in the form of characteristically function curves.

Use-case:

1. The user starts the method on the client-side and opens a ship project. He makes a connection to one or more accessible servers and from the visible projects the selects what is of relevance to him.
2. Then follows a composition of a simulation model consisting of local and remote methods. Methods which can be simulation methods of different simulation depth or methods for reading or editing locally saved simulation models.

[3] Flensburger Schiffbau Gesellschaft mbH & Co. KG; http://www.fsg-ship.de/
[4] SAM Electronics, http://www.sam-electronics.de
[5] Ansoft Simplorer; http://www.ansoft.com/products/em/simplorer

3. The remote components are copied to the local system.
4. The server connection is terminated.
5. The local components are being supplied with parameters.
6. The simulation calculation is started.
7. The complete simulation is evaluated.
8. The overall model contains a variant notation and a version number
9. The method is ended.

Simulation of ship and navigation equipment

Scenario supplied by FSG, SAM and Technical University of Hamburg-Harburg[6]

Already in the proposal phase and particularly in the design-concept phase it is important to test the interaction between the ship trunks, the engine, the steering controls and the track control. Particularly, the track control is individually adapted to the ship according to the steering controls. This is currently done by actually performing a test sail.

Unfortunately at this point it is then already too late to make changes to the trunk. By the use of simulations it is possible to test if all the components fit together before actual performing the test sail.

A concept engineer at the shipyard needs to test the manoeuvrability of a certain ship concept. To do this he will use the ship simulator SimFlex from Force (DMI)[7]. This ship simulator is available at both FSG and TU Hamburg-Harburg. The simulator is supplied with the data concerning the hydro-dynamic properties of the trunk, the engine and the steering controls. The simulator contains a track control test or is being connected with one. The simulation is started and the simulated results (coordinates) are saved for later use.

In a modification of the above scenario the simulation is performed at TU Hamburg-Harburg to test the manoeuvrability. The procedure is the same as described above.

In a further modification of the scenario is performed during the manufacturing of the navigation system for testing the regulations of a particular track control or for the preparation of the commissioning. In the ideal case the navigation system is delivered with a configured track control. More important that optimization is done before production.

Use-case:

1. The concept engineer chooses a project (ship, version, etc.)
2. The engineer starts the SimFlex simulator which runs as separate application on the same computer or on a computer in the network. The application accesses the data from the open project which are stored on a data server and thus are accessible from all of the involved companies.
3. The SimFlex application is not part of the SESIS system and needs to be available locally. It doesn't appear in the system as a separate component and is also not started by the system. Instead there is a method that supplies it with data from the system.

[6] Technische Universität Hamburg – Harburg; http://www.tu-harburg.de
[7] Force Technology / Danish Maritime Institute; http://www.force.dk

4. This method knows the source of the data; converts and prepares the data for use by SimFlex.
5. Beyond the description of the hydro dynamics, the engine and the rudder; the parameters and the filter characteristic curves of the track control are required as well.
6. The behaviour of the track control is simulated by the simulator by means of the loaded parameters and the filter characteristic curves. When SimFlex is not capable of fulfilling the task, other simulation software can be used or a track control simulator can be attached though the network.
7. The outcome of a simulation run consists of among others, simulated tracks which are then stored so that they can be accessed by other components in the system.
8. The concept engineer ends SimFlex and the corresponding component. The method writes the data and metadata in the system and frees the lock.
9. The concept engineer can now end the system or work further by using other methods.

Rapid prototyping / extending the system

Scenario supplied by FSG

A customer asks for a calculation/estimation that the current SESIS system cannot deliver. In order to deliver the inquiry nonetheless, a developer has the possibility to dig right into the system and develop new methods or change already existing methods.

Use-case: Extending/improving existing methods

The following steps are being performed in iterations until the developer is satisfied:

1. The developer starts a method in debug-mode and
2. He follows the run of the component stepwise until it reaches the point where something needs to be changed.
3. It is now possible for the developer to temporarily change variables and the application flow to find out what needs changing in the source code.
4. The method is ended.
5. The planned changes are made in the source code.
6. The method is translated by a compiler and it is added to the system, marked as a prototype.

Use-case: Adding new methods

In this use case a method that can be extended or changed doesn't exist, so instead a method will be written from scratch using the RCE Software Development Kit that is part of SESIS.

1.3 Thesis Description

The aim of the thesis is to make RCE Grid-aware and look into where Grid resources might be beneficial for RCE. As a result a conceptual design for using Grid resources within the RCE needs to be developed.

There are various places where using Grid resources can be beneficial, and as a minimum the component developers need to be able to use Grid resources without any hassle, especially when writing new or rewriting methods.

It is an additional requirement that all Grid support added to RCE needs to work with two different Grid middleware solutions, namely the Globus [Globus] and the UNICORE [UNICORE] Grid middleware.

It has been decided in the design phase of RCE that the version 4 of the Globus toolkit is to be used. This version is based on Web Services and the WSRF specification is used to make these web services state full. Web Services and WSRF are described later in this thesis.

All design needs to be UNICORE ready, but supports for UNICORE can be added at a later date. All implementation work, done in the period of this thesis will thus be done using the Globus Toolkit version 4. This decision has been made as only a Globus grid installation is at hand and because there are no plans for making a UNICORE installation for the time being.

Chapter 2

Grid Computing

In the following section I will look a bit closer at Grid Computing in order to make it clear what it is, what it brings the project and what we should have in mind when considering taking advantage of a Grid system. I will also look a bit at origins of Grid computing and at the current trends, as well as what the Grid could be in the future if we were not limited by time, money or standards.

2.1 Introduction

Grid computing is a computing model that provides a resource pool much like a power grid provides power. The best way to understand grid systems is indeed thinking about power grids. When we plug in equipment consuming power, we expect it just to work. The correct voltage should be available without us worrying about where the power source is located. Instead of each house having a power generator the power grid provides a *virtual generator* that adapts to the customers power needs.

The idea with grid computing is making a grid consisting of heterogeneous computer systems, thus creating the illusion that there is just one *virtual computer system* – or virtual computer resource pool.

2.2 Abilities

In order to understand why indeed such a virtual computer system is of use we will have to look at the different abilities a grid has to offer. The use of these abilities often characterized the grid type, though there in practice are no hard boundaries between the different types. The types people generally talk about are scavenging grids, computational grids and data grids.

2.2.1 Resource Utilization

Many organizations have many underutilized resources as most desktop system are rather idle and might only be busy 5% of the time. This grid computing model can improve upon this resource utilization, by offering the unused resources through a grid system.

The easiest way of using these resources is running tasks, or so-called jobs, on idle computers, thus talking advantage of free CPU-cycles and memory. In order for this to be applicable, the application needs to be able to be run remotely without overhead, which means that the remote systems need to fulfill any special hardware and software requirements imposed by the application. There should also not be too much overhead in moving the application and data to the remote system. A grid system utilizing idle resources from desktop computers is often named a "scavenging grid".

It is indeed possible to utilize other resources than CPU-cycles and memory and especially "data grids" makes use of the often enormous unused disk capacity by aggregating unused storage into one virtual hard drive. "Data grids" can also archive improved performance and reliability much like RAID systems.

2.2.2 Parallel Usage

It is clear that an application cannot use more resources than available on the computer it is executed. This is often a problem as some calculations are so demanding that they take ages to perform on even the quickest computer available today. One way to solve this is taking advantage of more processors and thus parallelizing the code path, so that sub calculations not depending on each others can be executed on separate processors in parallel.

If an algorithm can be separated into independent parts, it can be parallelized on the grid as well, by submitting the independent parts as sub jobs. It is not all gold and glory though, as not all algorithms are parallelizable, and even if they are they might not scale to more than a few processors. Also the overhead and latencies of transferring the data from the various subjobs has to be taken into consideration when parallelizing an application. Sometimes, it is not the algorithm that can be parallelized but the data, as amounts of data can be processed separately. Such data separation is scale free and with 10 computers, the processing will be finished in 1/10 of the time as with only one.

This parallel capacity is probably one of those things that really have encouraged the development of grid system as it makes it possible to perform huge computations that weren't possible before and thus have a huge impact on chemistry, physics, financial modeling etc.

Most often dedicated servers are used with the idea of setting aside resources for performing computations, and for this reason such grids are often named "computational grids".

To summarize: This grid architecture is able to distribute process execution (the jobs) across a parallel infrastructure and make use of unused resources. This provides the user with ability to deal with large data sets, as well as solve large-scale computational problems that are too complex for one machine to handle. The data sets can be split up into smaller parts, each which can be dealt with on a separate grid node and a parallel division of labor between processes can minimize the time needed to perform the a large-scale computation.

2.2.3 Collaboration

Grid systems also make it possible for people and organizations to cooperate by sharing virtual resources, such as hardware resources, data sources, etc. As a reason for this, grid computing supports managing restricted user access, security, resource reservation and accounting. It is then possible for a group of individuals or institutions to share the computing resources of a grid for a common goal, which has made the way for national research grids. Such a group is considered a *virtual organization* in grid terminology

2.2.4 Special Resources

There are a lot of other computer resources than just CPU, memory and storage resources, and it is also possible to use the grid for sharing these as well. This way it is possible to share printers, software licenses etc., though sometimes special software (grid jobs and services) will have to be written.

For instance a company might have just a few licenses of software to generate PDFs from Word documents. In this case an application could be written that would locate a grid node with the software installed and then transfer the word document to the machine and transfer the PDF file back.

2.2.5 Other Abilities

Though the above abilities are probably the most used ones, there exists many more. The ability to suspend a job and resume it makes it possible to balance the use of resources, by moving the job to another node and then resume the execution. This also offers great flexibility and reliability as a job for instance can be moved if a computer needs to be taken down for maintenance.

2.3 A bit of History

Grid computing is around 10 years old and it all started around the time of the Supercomputing '95 conference. Two weeks before and during the conference, the director of the mathematics and computer science division at Argonne National Laboratory[8], Rick Stevens, suggested establishing a link between 11 research networks to form a national-wide Grid, the so-called I-WAY [IWAY].

A small team led by Ian Foster at Argonne, developed new protocols which allowed the users of the I-WAY to run applications on computers across the country.

A small team led by Ian Foster at Argonne then created new protocols that allowed I-WAY users to run applications on computers across the country. The experiment was successful and gained founding from the Defence Advanced Research Projects Agency (DARPA) and as a result the first version of the Globus Toolkit [Globus] was releases in 1997.

The Globus Alliance has since then expanded and includes many cooperative partners and sponsors such as DARPA[9], NASA[10], IBM[11] and Microsoft[12].

At the time of the release of The Globus Toolkit, the development of UNICORE [UNICORE] was initiated in Germany in order to provide the users of the German supercomputer centres an integrated Grid middleware solution that was an alternative to the Globus Toolkit. At first

[8] Argonne National Laboratory; http://www.anl.gov
[9] DARPA; http://en.wikipedia.org/wiki/Defense_Advanced_Research_Projects_Agency
[10] NASA (North American Space Agency); http://www.nasa.gov
[11] IBM; http://www.ibm.com
[12] Microsoft Corporation; http://www.microsoft.com

a prototype was developed, but the foundation of what is UNICORE today was developed in the follow-up project UNICORE Plus which ran from 2000-2002.

2.4 Grid Systems today

Grid computing is a fairly new technology that in many ways can be seen as the next logical step in distributed networking. It is clear that it can bring many advantages when dealing with large data sets and with computational intensive computations, but as with everything else there are downsides.

The current situation is that there are almost Grid systems everywhere, with huge Grid systems here in Europe such as D-Grid [D-Grid] and CERN[13] and there is quite some applications taking advantage of these systems.

But there are also problems. One of these is that there exists a manifold of different Grid systems and it is quite a task for a developer, who is new to Grid computing to learn about all these various systems, their differences, their advantages and disadvantages and their future prospects. Due to the "lack" of standardization you often get locked-in with the system that you have chosen. This is enough for most to keep their hands away from Grid computing if they can avoid it.

As mentioned before, there exists many different use-cases for using grid technology, and most of the grid middleware solutions out there are developed with one major use-case in mind, such as data-grid, computational grid etc.

Things are changing a lot though. The developers of grid systems have realized that is it possible to separate the basic components of grid systems into replaceable services, which means that instead of changing the grid middleware it should be possible to change a service instead or add additional new ones. Such a service could for instance be a different scheduler, or a service making a database available on the grid[14]. It should be kept in mind that you can access these services from your grid jobs.

To avoid vendor lock-in, and to make it possible for grid developers to write services that can be used by different kinds of grid middleware, more than replaceable services are needed. More generally, the interfaces between these services needs to be determinates, as well as which services are needed to have a fully functional grid environment.

2.4.1 Open Grid Services Architecture

In the paper "The Physiology of the Grid" [Physio], people from Globus, IBM[15] and two American universities proposed the Open Grid Services Architecture[16], which is an initiative

[13] CERN; http://public.web.cern.ch
[14] Such a service already exists and is called OGSA-DAI (http://www.ogsadai.org.uk/). The difference between accessing a database from a job though OGSA-DAI instead of the native method is that the grid takes care of things as resource discovery and rights management. The grid service also has the ability to limit data movement by accessing a replica database closer to the grid node running the job requiring database access.
[15] IBM has a commercial grid middleware product based on Globus under the name *IBM Grid Toolbox*; http://www-128.ibm.com/developerworks/grid/library/gr-develop/
[16] OGSA - The Open Grid Services Architecture; http://www.globus.org/ogsa/

to determine the services needed for a fully working grid environment as well as how they interact.

It was realized that even though grid services have started far apart in application domains and technology of web services, they share a lot of similarities with web services. Because of this and because of the maturity of web services, a lot of effort has been done reusing this proven technology and extending it in the ways needed.

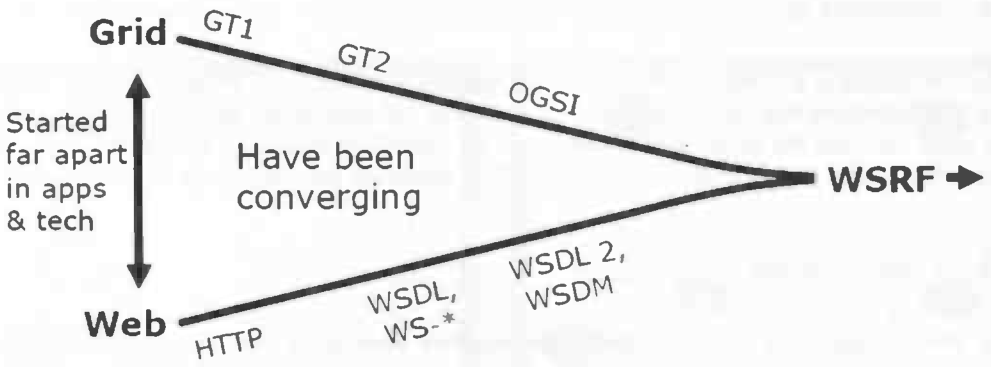


Figure 2: Converging of Grid and Web Technologies
Reference: Alexander Reinefeld: Grid Computing (slides)[17]

OGSA has been well received and a lot of work has been done to realize the interoperability of the two most deployed Grid middleware systems, Globus and UNICORE by adapting OGSA for UNICORE as well.

Most of this work has been done in the GRid Interoperability Project (GRIP)[18] which was funded for 2 years by the EU and the follow up project UniGridS [UniGridS] that is still running to the end of this year. The aim of the last project is to develop an OGSA[19] compliant Grid Service infrastructure based on UNICORE by for instance adding support of the new Web Service Resource Framework [WSRF] standard developed by the Globus Alliance and already integrated in the Globus Toolkit 4.

2.4.2 Web Service Resource Framework

Web services as such are stateless, which means that they retain no data between their innovations. In order to fix this particular problem the standard WSRF [WSRF] has been developed.

In short, WSRF is a set of specifications designed to merge Grid and Web technologies by framing the concepts of the earlier Open Grid Services Infrastructure [OGSI] in terms of current Web Service standards such as the Web Services Description Language [WSDL].

[17] http://www.dini.de/veranstaltung/jahres/2004/vortraege/DINI-Reinefeld.pdf
[18] GRid Interoperability Project; http://www.grid-interoperability.org/
[19] OGSA - The Open Grid Services Architecture; http://www.globus.org/ogsa/

Basically what WSRF does is provide a set of methods that stateless web services can implement to become stateful, which is needed by Grid services. It does this by letting the web services communicate with resource services that allow data storage and retrieval.

WSRF support is currently implemented in version 4 of Globus, and work is being done on making UNICORE use WSRF as well [UniGridS]. Since the UNICORE implementation isn't fully complete all UNICORE support to RCE will be added at a later date.

2.4.3 Summing up

Grid Computing is a relative new technology and major changes are happening from time to time. Some believe that the standardization work will still take years, also due to competition and pride between the major grid players, and as a result of this more and more client-side abstraction layers pop up such as GPE and GAT[20], which are described later in this thesis.

2.5 Future vision

It is always interesting and scientific when you free your mind from how a technology is realized today and what it has become and instead look at what it could be if we were to implement it today, 10 years later.

After having worked with and done research about grid systems during this thesis period, the more convinced I have become that the current grid middleware systems are highly capable of doing the things they have been designed for.

Standardization and use of proven web service technology it is becoming easier and easier writing grid jobs and grid services and also interchanging different grid middleware systems.

As always there are things that could be different. One problem with grid systems is that some software imposes hardware and software requirements that prevent the jobs from being run on all grid nodes available. But with the increasingly use of new portable programming languages such as Java and C# this is now less of a problem.

One more serious problem is that rewriting an application to take advantage of a scavenging grid or grid systems in general, is a reasonable sized task which in return makes the application highly dependable on grid middleware.

What I eventually would like, would be a system where standard application threads could be distributed to other computers connected in the grid automatically, if that brings performance benefits or improves resource utilization.

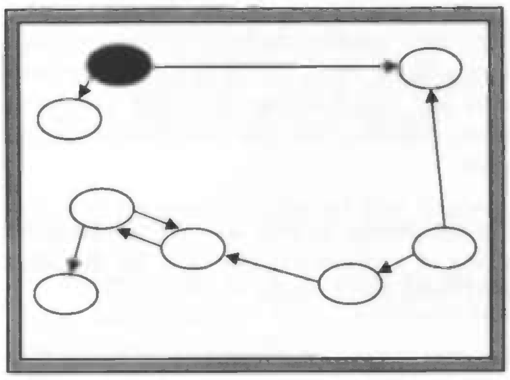
Research is being done in this area and there exists an open source project aiming at making this possible by introducing so-called active objects.

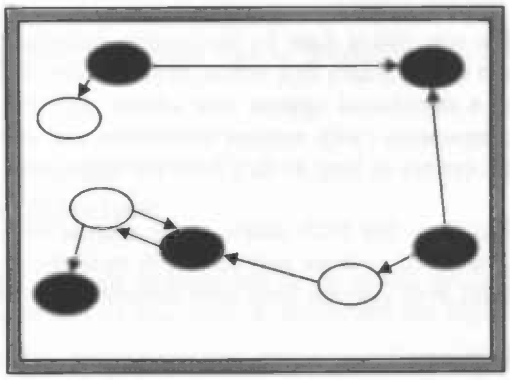
The product called ProActive[21] consists of a Java library that introduces the active object model original introduced by the Eiffel language.

[20] More on these abstraction layers can be found in chapter 5.
[21] http://www-sop.inria.fr/oasis/ProActive

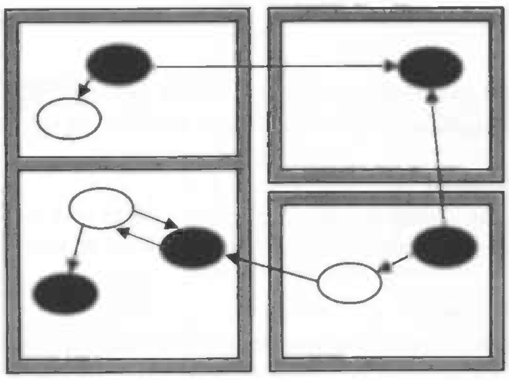
The idea is that the application is structured in various subsystems each with one so-called active object signifying one thread. This was each subsystems consists of one active object and various passive objects, if any, that are not shared between any subsystem.
This construction is interesting as only the active objects are known outside of the subsystems which allows for distributing the subsystems onto grid nodes.

A normal sequential program looks like below. The block object signifies the active object and the gray box signifies a computer host. As program like this can be multithreaded by making other objects active – just as long as no passive objects are shared.


Sequential program


Multithreaded program

By using this model, implemented by ProActive it is now possible to distribute one or more of these active objects (and associated passive objects) to different grid nodes.


Distributed program

This is interesting as it is possible to write your application with threads in mind and later simply convert these threads into grid jobs without making the code grid specific. This means that standard thread enabled applications can take advantage of grid resources very easily which make the use of scavenging grid systems very interesting in companies.

2.6 RCE and Grid Computing

In this section I will look at how the architecture of RCE fits in with Grid technology.

2.6.1 Similarities and Differences

Recall that SESIS is a project to develop a concept and simulation system for building ships with the main objective to make it easier, quicker and cheaper for the shipyards engineers to develop/design new ships.

The way this is done by performing complex collaborative simulations between the shipyards and the suppliers in a virtual organization. SESIS or actually RCE, can in this regard be seen as a distributed system that allows different users and organizations accessing and using components (with various simulation and calculation features) and data stored anywhere on the system as long as they have the appropriate rights.

It is clear that RCE shares many aspects with Grid computing, as they are both fundamental distributed systems and they both need the managing of a virtual organization. On the other hand, both systems have been designed with very different uses in mind.

RCE supplies a system for cooperating with other companies and suppliers by sharing data and methods (components) for calculation and simulation; whereas a Grid computing supplies a system for sharing and optimizing the utilization of computational and storage resources.

Clearly, RCE fulfills a different role than a Grid system and therewith justifies its existence.

2.6.2 So where does Grid Technology fit in?

One of the advantages of Grid computing is that it makes it possible to actually perform very computational intensive computations and simulations as well as making it possible to share computer power across organizations.

It is of interest to the future users that RCE has the ability to take advantage of this and as a minimum make it possible to write/rewrite simulation and calculation methods so that they make use of Grid resources.

There might be other areas where Grid support in RCE makes sense, but we will look into that later.

2.7 A Closer Look at RCE

Before I get too detailed we will have to look at bit closer at the RCE architecture that supports the SESIS system. We will do this by looking at the system architecture.

The ground principle behind the RCE architecture is that:

Every computer in the distributed (or local) system should contain at least one installation of the basic software. Depending on the kind of system (client, server, etc) this installation might have another configuration (extra plug-ins, etc).

To put it more simply, this means that each computer will always contain the same basic system (The SESIS basis system in the case of SESIS), which by the use of additional plug-ins can be adapted to specific tasks (data server, GUI client, calculation server, etc).

The advantages of this model are:

- Homogeneity of all systems
- The installation and updating of the software is always consistent
- The communication protocols between the installations can be predetermined and adapted to the network infrastructure
- No central services are needed

2.7.1 A Layered Model of the System Architecture

The layered model forms the basic for the overall software architecture. In the model the data, design and logic have been completely separated and in this way, it represents the logical boundaries in the system.

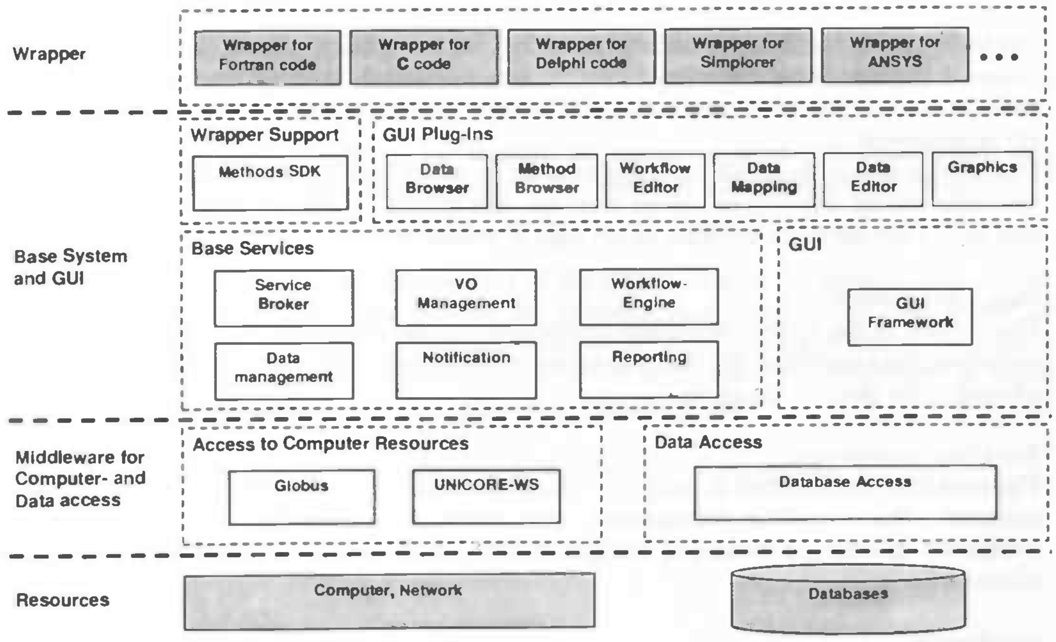


Figure 2: Layered model of the system architecture

The Resource Layer

The resource layer contains the hardware and operating system software needed to run the system. This layer is logically responsible for executing processes and saving data.

The concrete products in this layer (hardware manufacturer, database vendor etc) have not been determined as it can be abstracted by the above Resource-Middleware layer.

The Resource-Middleware Layer

The following layer is responsible for access though the resource layer. The access to the computers and the data storage is made available here through the use of interfaces. Only by the use of these interfaces it is possible for the components of the above layers to gain access to the computer resources and data storage.

The two parts of this layer are:

Grid middleware and abstraction
As it is a requirement that the system is Grid compliant, the access to computational and network resources need to use some kind of Grid middleware. This layer needs to support both the Globus and the UNICORE Grid systems.

Data storage and access
This part abstracts the access to data storage (database etc). In particular this part maps the data structures from the system to the structures stored in the data storage (tables in the case of relational databases).

The Basic-System Layer

The basic system contains the necessary services for the running and the logic of the overall system. It consists of the following:

VO management
This service is used by all above layers in order to grand the logged-in user rights to perform particular actions. For instance, some users are only allowed to execute particular procedure if they have a license to the software or the right to access the data.

Data management
This service is the system dependent component for accessing data. In contradiction to the underlying data interfaces, the data management offers more abstract interfaces that have been adapted to the goals of the system.

Workflow management
The workflow management is used to work out the order in which the procedures need to be executed. The workflow management also contains functionality for configuring the workflows (saving and loading of workflows, etc) which then can be used from a workflow editor GUI.

Notification
This service delivers notifications from the whole systems to the users. The notifications are ordered in categories.

Service broker
This service delivers references to components in the complete system environment. It is a decentralized service in which the data are organized in a hierarchic structure.

Reporting
This service makes it possible to manage information from the components. The information can be exported, printed or real reports can be generated.

The Graphical UI Framework

The Graphical User Interface layer serves as the basic user interface for the basic system and it can be extended by the use of bundles. The user interface is built on the Eclipse [Eclipse] Rich Client Platform.

As an example, SESIS includes the following bundles: Data Browser, Procedure Browser, Workflow Editor, Data-Mapping Editor, Data Editor, Graphical Output and User and Right Management.

Wrapper and Methods

The wrappers are the adaptors between the overall system and existing application code that are responsible for a job of a specific domain (engineering terms, mathematical, technical, etc). The adaptors are needed as some users have existing source code or full-blown external applications for fulfilling a particular task, that they need to access from within RCE.

With the use of different kind of wrapper technology the existing code or applications can be integrated with and used from RCE.

2.7.2 The overall System Architecture

RCE is a bundle/component based architecture. By means of the bundles the system can easily be extended. Each installation will normally consist of a different number and kind of bundles.

In order to insure some kind of homogeneity of the overall system, all systems will contain certain bundles, which then make part of the so-called base system. The base system is realized by using the Open Service Gateway initiative [OSGi]. This specification describes a component framework for Java, where bundles (which is the name of components in OSGi terminology) can be installed, started, stopped and removed on the fly.

The following figure gives a rough overview over the design of the system.

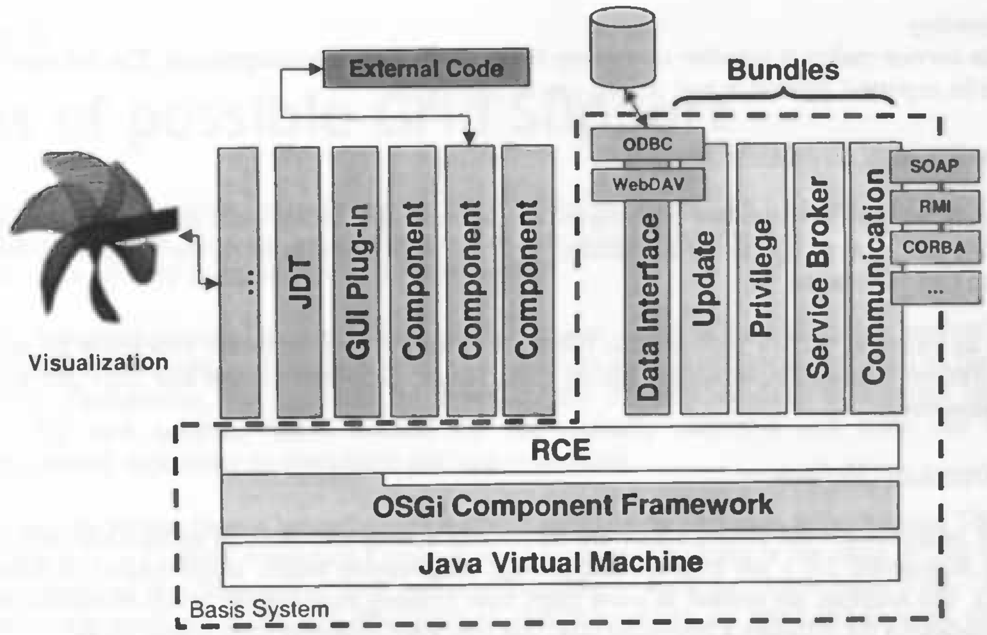


Figure 1: The architecture of the RCE, incl. the basic system

To elaborate a bit on the above figure, I will shortly explain the major building blocks, and later I will take a look at the layered model. The basic system consists of the following building blocks:

Java Virtual Machine (JVM):
The Java Virtual Machine serves as the interface between the system (Windows, Linux etc) and the underlying hardware.

OSGi component framework
The implementation used of OSGi manages the *local* bundles (OSGi components) of the system. The different states of the bundles and further information of the bundles are handled here.

RCE (Reconfigurable Computing Environment) layer
This layer manages the bundles in the distributed system. Additionally, it also takes care of the security infrastructure.

Privilege Bundle
This bundle is for managing the user rights and privileges throughout the system. When the system is started the user needs to identify him-/herself and during the user session. This bundle takes care of managing the privileges.

Service Broker Bundle
The references to the separate bundles are queried by the use of this service. This system is build after a model that is close to that of the internet DNS system.

Update Bundle

New versions of parts of the base system or other bundles are distributed by the use of this service. It offers the possibility to update the base system and its extensions.

Communication Bundle
The communication in the distributed system is realized by the use of this bundle. All communication is thus centralized in this bundle, which offers support for various communication standards such as CORBA, RMI and SOAP.

Chapter 3

Areas of possible Grid Support

Earlier I looked at what I actually would like the Grid to be and how I would like to use it. Unfortunately, the ideas are out of the scope of this paper, thus much can be learned from them when actually designing the RCE Grid support.

In the following sections I will look at the various options available for adding Grid support to RCE, and I will evaluate these by looking at advantages and disadvantages for each of these options. Furthermore, use cases will be shown for the different solutions as different ones exist for each solution. These options are not mutually exclusive and more can be implemented, depending on usefulness and time constraints.

The options results from a brainstorming meeting between me, Andreas Schreiber and Thijs Metsch here at German Aerospace Center. Later I have looked closer at the ideas to find the advantages and disadvantages of each option and in order to see if they made sense at all. This chapter represents my analysis of these options. First the options are listed and grouped, then the advantages and disadvantages are discussed and later I look at use cases for each option group. After the analysis one of the options is chosen with an explanation why.

3.1 The options at hand

As a *minimum requirement* we need client-side Grid support (Option 1), which means that we simply need the ability to use Grid resources from RCE-procedures by starting external applications (executables) on the Grid. There are two options for this which will be described later on.

Grid-aware Method API

1 a.	Support added to the plug-in SDK
1 b.	Support added as Globus/UNICORE bundle

The other options for adding Grid support are server-side, i.e. meaning that we are talking about actual Grid Services. The first option is deploying the RCE instance as a Grid Service (Option 2), which we can do the following way:

Deployment of the RCE instance as a Grid Service

2 a.	Starting RCE by the use of a Web/Grid Service
2 b.	Deployment of the complete system
2 c.	Deployment of the interfaces of the RCE layers

The last option is simply deploying the individually RCE bundles as Grid Services (Option 3).

Deployment of individual bundles as Grid Services

3 a. Deployment of each bundle as a separate Grid Service.

3.1.1 Option 1: Grid-aware Method SDK

This solution is a requirement for RCE and it is also the most interesting as it adds the ability to actually take advantage of the Grid when developing new components/bundles for solving particular calculation and simulation problems.

The solution represents one of the most common usages of grid systems; to request resources and start jobs on the grid.

The difference between solutions 1a and 2b are minor, as 1a requires the method developer to manage the host names and ports manually either by hard coding in the source code or reading from a configuration file, where as solution 2b takes care of all this by having the configuration in a separate bundle/component.

3.1.2 Option 2: Deployment of the RCE Instance as a Grid Service

Before I go further I need to explain the difference between grid services and web services as it is not clear to most readers.

In brief, a grid service following the WSRF standard is a WSDL [WSDL] (Web Services Definition Language)-defined service that conforms to a set of conventions related to the interface definitions and behaviors. This means that each grid service is a web service where as the opposite is not always true.

Grid systems are service based, meaning that the functionality offered is based on services. For instance there might be a service responsible for file transfers as well as another service responsible for job execution. In order to use proven technology and get the advantages of this, the trend is using extended web services instead of self invented technology. This brings some advantages like that it is possible to write normal applications or web applications using these services using standard tools; resulting in greater interoperability.

But basically, grid services are just services offering "traditional" grid capabilities using the standards of Web Services and not more than that; for instance writing a grid service doesn't mean that the service "runs on the grid".

The advantages for RCE if I implement this option is that it is possible to start RCE by the use of a grid service and that we can offer the RCE API through a Grid Service. The latter meaning, that application developers can use this non programming language specific API for integrating RCE with their applications. This also means that it is possible writing a web interface to RCE.

The question is how much this benefits us. Regarding option 2a it does not seem to be a solution worth pursuing as it will be possible to start an RCE instance by the use of SSH, which will even be more efficient, and which is just as easy. The downside to using SSH, is that you need a SSH server installed. On the Linux platform this is basically not a problem as most Linux distributions already come with a SSH-server, but if you are running SESIS/RCE on Windows you will have to either purchase a commercial SSH-server or use the Windows port of OpenSSH[22]

With regard to offering a web serviced based API to RCE; we have two options, option 2b and 2c. Deploying the whole system as a Grid Service (option 2b) is not really feasible due to performance and load problems. This is especially due to the fact that the system needs to remain scalable, which we cannot guarantee because the users, out of our control, can add heavy plug-ins which will be hosted in the web container as well.

An alternative to this is to not actually deploy the whole system as a Grid Service, but instead give RCE a WSRF interface. This way we won't experience the same performance and load problems as with option 2b. The proxy could possible be generated with the use of introspection[23], but could also be written manually as the API should remain stable for a foreseeable future.

3.1.3 Option 3: Deployment of RCE bundles as Grid Services

This option has some fairly big disadvantages that need to be leveled by the advantages as the way the RCE layer talks with the bundles needs to be changed.

The solution is a bit similar to option 2b, but it has the advantage that each bundle will run as separate services which will make the load problem less of a problem. On the other hand there might be a performance penalty as each grid service is a web service, which has a lot of communication overhead due to the use of SOAP.

If this solution is chosen, the RCE layer needs to talk to the bundles using networks communication through the use of web service standards such as SOAP, instead of talking directly as in the current design. The way the communication bundles talk to each other also needs to be redesigned.

All this adds an extra overhead to RCE compared to the current design of RCE, and it is hard to see what this really gives us, unless we wants parts of the basic system or the third-party plug-ins to run on other machines. It is possible to implement this, but big parts of the RCE design needs to be rethought.

On the other hand third-party procedures can already run on other machines, as long as the basic system is installed. All communication will then be through the Communication bundle.

Additionally, extension method developers would have to write a web service for each method, which will make it harder developing these.

---

[22] OpenSSH for Windows; http://sshwindows.sourceforge.net/

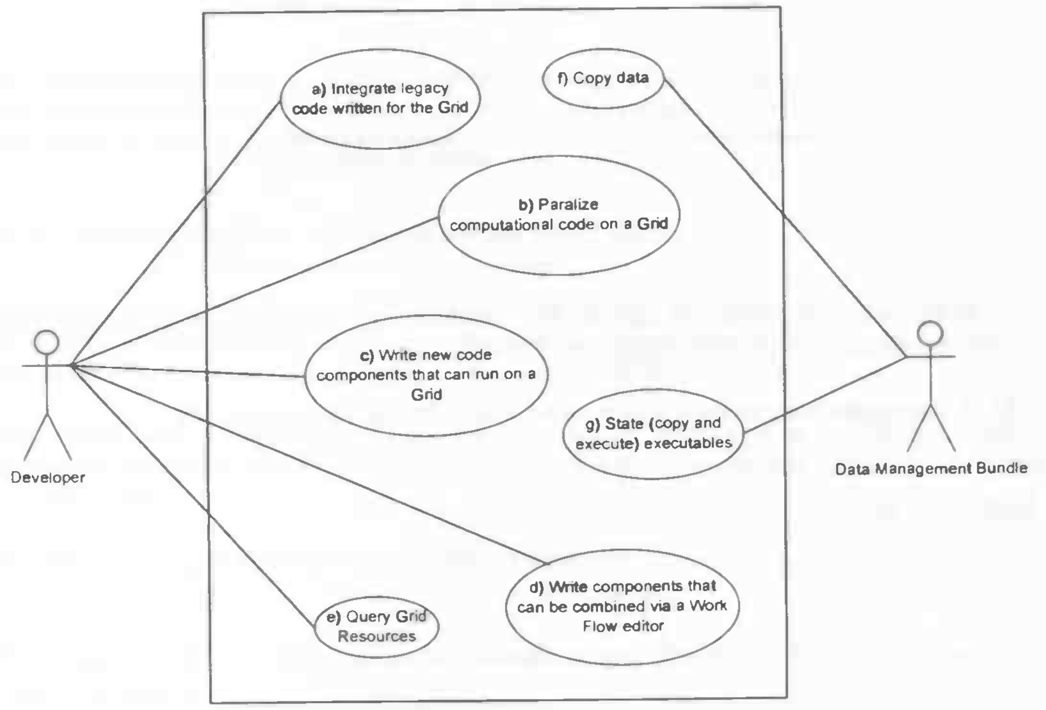
[23] Introspection (self-examination) is a technique that refers to the ability to examine something to determine what it is, what it knows, and what it is capable of doing. Languages such as Python and C# have the ability to introspect objects etc at runtime.

3.2 Use Cases

In the following section I have modeled the requirements by the use of coarse-grained use-cases, to get a better idea of what makes most sense to implement. This has allowed me – in cooperation with the German Aerospace Center - to make a decision.

3.2.1 Option 1: Grid-aware Method SDK

For the SDK there are various use cases for adding API that makes it possible to make grid-based methods. The most important ones are listed below and further elaborated.


a) Integrate legacy code written for the Grid
f) Copy data
b) Paralize computational code on a Grid
c) Write new code components that can run on a Grid
g) State (copy and execute) executables
Developer
Data Management Bundle
d) Write components that can be combined via a Work Flow editor
e) Query Grid Resources

a) Some of the potential users/companies of RCE have legacy code for performing calculations and simulations. Some of this code is written for the grid and integrating this with RCE as RCE bundles/plug-ins is a requirement.

b) The potential users have source code that can take advantage of the parallelism of a grid and need an SDK that proves useful getting the code to run on a grid.

c) Third-party developers want to write bundles/plug-ins as separate method that can run on a grid and be combined in various ways using a Work Flow editor.

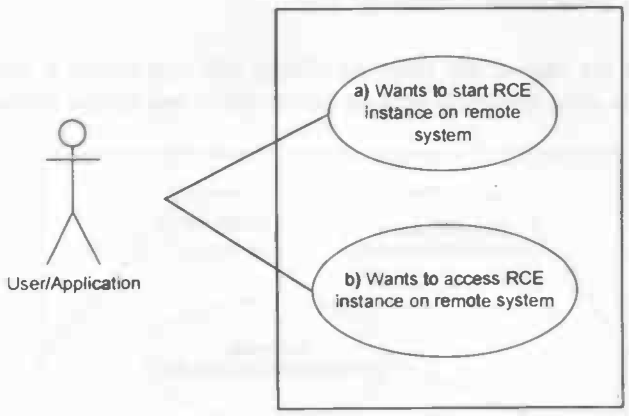
d) The developer wants to query the grid for available resources (CPU time, etc).

e) The Data Management bundle needs to copy required data to a grid point.

f) The Data Management bundle needs to copy an executable to a grid node and when copied, execute it.

3.2.2 *Option 2: Deployment of the RCE Instance as a Grid Service*

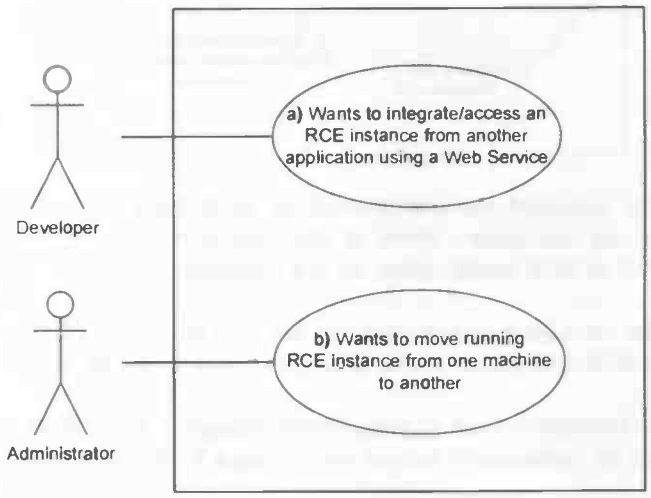
<u>Solution 2a</u>


a) Wants to start RCE instance on remote system
b) Wants to access RCE instance on remote system
User/Application

a) A user needs the ability to start an RCE instance on a remote system. An application communicating with RCE might need this ability as well.

b) A user/application wants to access an RCE instance on a remote system.

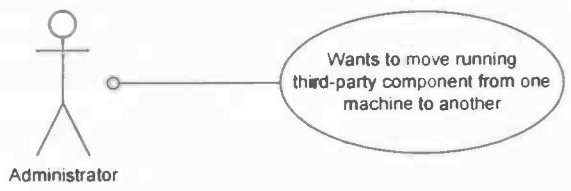
<u>Solution 2b and 2c</u>


a) Wants to integrate/access an RCE instance from another application using a Web Service
b) Wants to move running RCE instance from one machine to another
Developer
Administrator

c) A developer wants to develop a specialized application that uses/communicates with an RCE instance. A common way to communicate with remote software instances today is using web services.

d) An administrator needs to replace/shut down a computer with an RCE instance currently in use. The administrator needs the option to move the instance from one system on the grid to another transparently to the users of the system.

3.2.3 Option 3: Deployment of RCE bundles as Grid Services


Wants to move running third-party component from one machine to another
Administrator

An administrator needs to replace/shut down a computer with third-party methods currently in use. The administrator needs the option to move these method instances from one system on the grid to another transparently to the users of the system.

3.3 Chosen Option: Grid-aware Method SDK

The above use-cases show that there is much to be gained from implementing grid support in RCE. The second and third options are the most technologically interesting options, but the use-cases in the first option lays closer to the real need of the users of RCE.

These users need the ability to execute jobs and sub-jobs from within an extension method in a way that integrates with RCE and that is transparent to the underlying grid system (Globus or UNICORE).

For this reason and with agreement from the German Aerospace Centre, I have chosen to concentrate on developing a grid-aware method SDK for RCE.

More detailed the below-listed parts will be implemented. The design will be described in the following chapter.

- The possibility to use grid resourced from RCE, using the standard RCE proxy certificate as long as you have been granted right for doing so.
- The possibility to describe and run jobs with a Java-based abstraction API that fits in with the RCE framework. This included handing of file staging (copying input and output files between client and grid host), as well as output/input handling.
- The possibility to transfer files between grid nodes (including client) without actually executing jobs.
- The possibility of resource querying to insure that a host fulfils the requirements of the job.
- The option to implement another backend of the API so that other grid middleware systems can be supported (Other versions of Globus, UNICORE, etc)

Chapter 4

Requirements and Design

In the following section I will look at the requirements needed for the chosen option. The requirements for the chosen option were modeled in the previous chapter by the use of coarse-grained use-cases. The design of this option will be modeled in this chapter by elaborated use-cases and class diagrams following the UML standard.

Since the grid SDK will be based around grid jobs, I will first look at bit closer at these and how the work.

4.1 A look at Grid Jobs

A grid job is not necessarily a whole application, but more often a single unit of work within a grid application. A job usually requires input data and returns output data, as well as a particular execution environment, imposed by the choice of programming language, libraries used etc.

4.1.1 Job Criteria and Consideration

There exist various types of jobs that can be used on a grid system. Batch jobs are good candidates for grid usage as they require no user interaction and are often simple in nature. It is also possible to run standard applications on grid nodes such as for instance a commercial video rendering application. These can be a bit more complicated as they often require special operating system conditions and a manual installation procedure.

Of course, the best candidate for grid jobs are application that are parallel in nature, when then can be split into multiple sub jobs to be run in parallel or to be connected in a workflow.

4.1.2 Job Submission

Job submission consists of three stages. The first being staging of input, which basically means sending input data and possibly an executable to a grid node. The second is actually executing the grid job and the last stage is retrieving the results.

Since the machines are heterogeneous, staging an application might be troublesome as the application has specific operation system needs, as well as requires various libraries to be present. The way this is normally solved is not staging the actual application, but installing it on some of the grid nodes, or alternatively accessing it via a mountable networked file system; the latter still requiring a specific operation system and hardware architecture in order to work.

4.1.3 Programming Language

Grid jobs can be written in any language executable by the grid nodes where it is to be run, though it is an advantage using a language such as Java or interpret languages such as python as these are more portable and impose fewer requirements on the grid notes.

Some times a job needs to interface the grid infrastructure (grid services) and in this case, it is important that binding are available for the language used. With the introduction of web services, it is possible to interact though the web service interface which there exist support for, for almost all major programming languages.

In case of Java it should be noted that the name of the executable to run is "java" (the Java Virtual Machine) and the application will have to be given as the first parameter.

4.1.4 Data Input and Output

Input and output handing can be troublesome. Most batch jobs need some way to send error messages and warning to the user and some even require "console like" user interaction.

It can be even harder with real applications like for instance a commercial application. In order to interact with such a program a web service needs to be written that interacts as glue between the user interface and the grid job. The way it works is that the grid job exposes the interface of the application as a web service interface that can then be interacted with remotely using web service technology. Often a web application is build for this.

4.2 The Design of the Grid-aware Method SDK

In the following section I will look closer at the chosen option and by the use of more detailed use-cases I end up with an UML based class design of the public Grid-aware Method SDK.

Additionally the public API will be textually described and the class diagrams will show how the Globus support is implemented.

As mentioned in the introduction to grid computing, there exist different kinds of grid systems. Both the Globus and UNICORE middleware solutions have been designed with high performance computing in mind and can thus more and less be labeled "computational grid systems". The grid use cases that I have determined for the RCE platform are also of computational art, and the design of the API will thus follow this tradition.

In the previous section I described some general use cases for having client-side grid functionality in the SDK, which is to be used then writing new extension methods. In order to design such an API we need to look at the individual use cases needed to fulfill these needs. While doing the API design, the ideas from the section "Future vision: what the grid could be" have been in mind.

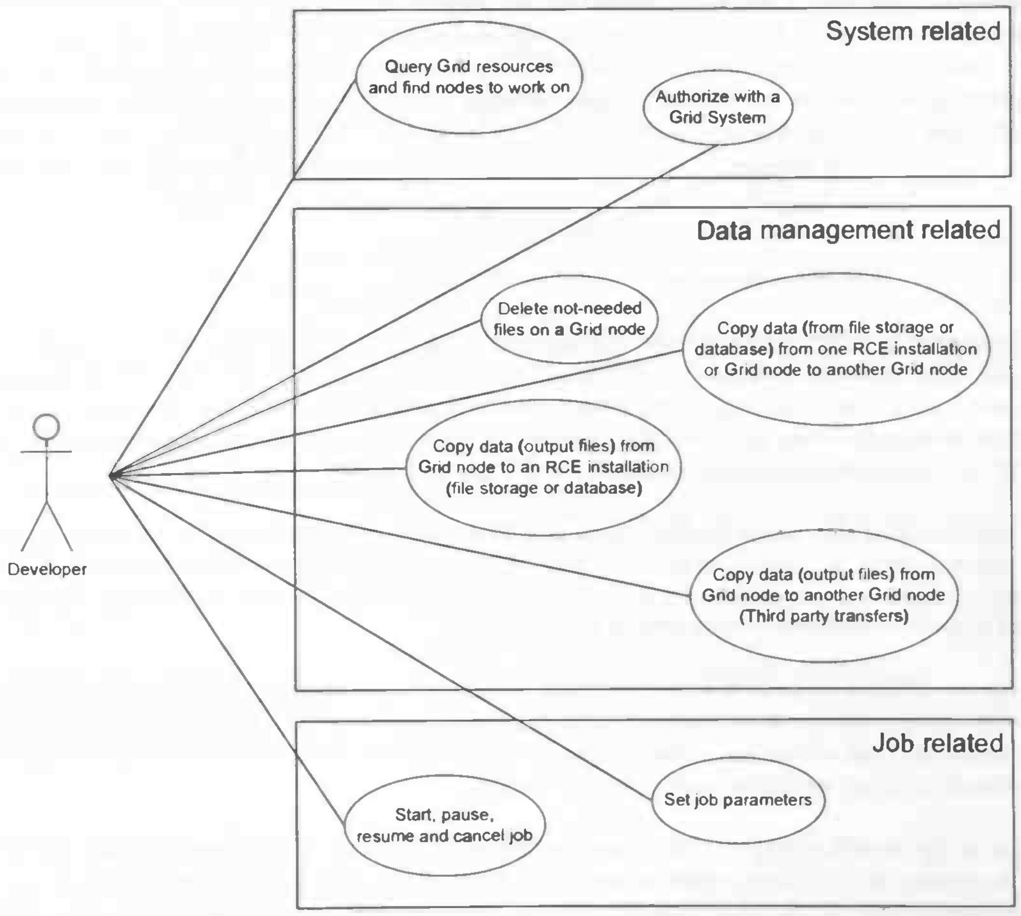
The SDK is the Software Development Kit, which will be used when a developer writes a third-party bundle – a so-called extension method - for the RCE system, as described in one of the case studies.

The idea is offering a simple API that integrates well with RCE. The API needs to be simple, yet offer the most common functionality needed by the procedure developers. In case the developer needs more functionality, it is possible to use the API for the grid system in use directly, even though this should only cover advanced use.

A typical scenario for the use of a grid resource is starting an executable on the grid, and is as follows:

1. Get certificate to be allowed to work with the grid
2. Specify Resource Requirements
3. Query the grid for resources
4. Transfer input files needed by executable (host etc)
5. Transfer executable
6. Run executable
7. Transfer output files

Based on the above I can describe the use-cases needed.


System related
Query Grid resources and find nodes to work on
Authorize with a Grid System
Data management related
Delete not-needed files on a Grid node
Copy data (from file storage or database) from one RCE installation or Grid node to another Grid node
Copy data (output files) from Grid node to an RCE installation (file storage or database)
Copy data (output files) from Grid node to another Grid node (Third party transfers)
Job related
Set job parameters
Start, pause, resume and cancel job
Developer

The use case diagram shows that we for the client-side grid support needs API in the following areas:

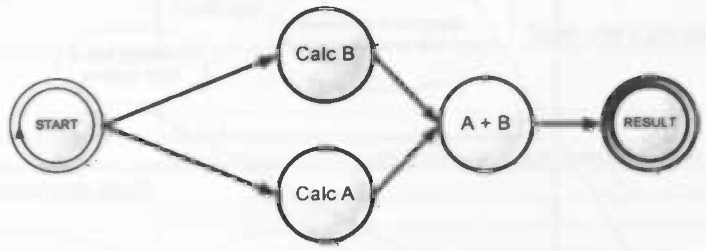
- System related (Authentication, Resource Description and Resource Discovery)
- Job related (Job Description, Submission and Handling of Input and Output)
- Data management related (Staging of Data Files)

Based on the above grouping I at least need separate classes for these areas of the API.

4.2.1 Workflow

Apart from the above the users of grid systems often need the ability to define workflows. A workflow can be defined as "*the computerized facilitation or automation of a business process, in whole or parts*", which basically means that the user want to be able to define a set of specific tasks (for instance grid jobs) and their relation.

Such a relation can be described in for instance a directed cyclic graph. A simple workflow could be: Calculate A and B and when both are calculated, calculate the sum, as shown below.



Transferring this to grid jobs, and lets say that "Calc A", "Calc B" and "A + B" were three grid jobs, then the workflow engine would have to run "Calc A" and "Calc B" on two grid nodes, which includes staging the executables and data needed. The engine would then also have to monitor when both jobs are finished and transfer the output data files and the job "A + B" to a grid node, and later transfer the result back to the user.

Implementing this seems feasible given an API exists for staging data files and running jobs, which is what is being developed as part of this thesis. Additionally, a description format would have to be developed for actually describing the actually work flow graph, consisting of elements (here grid jobs) and their dependencies.

On the other hand there is a lot to be gained from using an already proven workflow engine, as work flows often can be optimized by manipulating the graph. Graph theory is a big field within the field of computer algorithmic, and it is unlikely that I can do a better job than the already existing workflow engines on the market.

Since the workflow engine will be used generally for methods in RCE and not only grid jobs, the choice of workflow engine is not in the hand of the author of this thesis, but it can be mentioned that the Karajan Workflow Engine from the CoG toolkit (look at section 5.1.1) is a likely candidate. For this reason no workflow engine will be implemented as part of this thesis and I won't go into further details regarding grid integration and workflows.

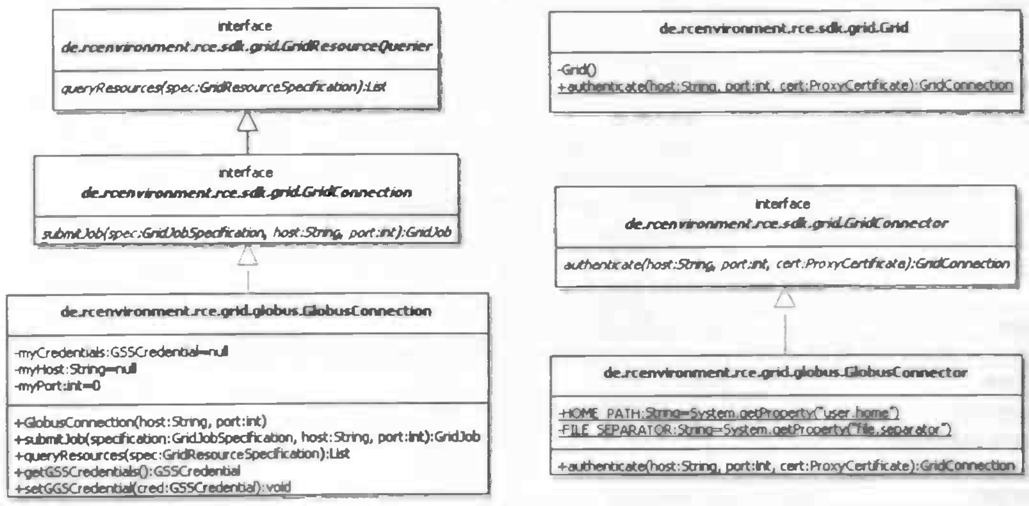
4.2.2 Grid Access and Authentication

Before being able to use an installed grid system, we need to obtain credentials in order for the grid system to know that we have the appropriate rights to access and use the resources available. For this reason I need a way to authenticate with the underlying grid middleware using a so-called proxy or certificate.

The RCE system already contains a so-called `ProxyCertificate` for each user which can be converted into something usable for authenticating with the grid middleware. This can be done by extracting an `X509Certificate` from the `ProxyCertificate` and modifying it.

I represent a connection to a grid by the `GridConnection`. Both querying (which will be dealt with later) and authentication have been put in this object and it uses two objects for implementing this which are described by two interfaces, `GridConnector` and `GridResourceQuerier`.

Below you see the public API for grid access and authentication which is represented green and the Globus implementation represented with blue. The user uses the `Grid` class to authenticate with the grid and get a `GridConnection` in return that can be used for querying resources and submitting jobs. Notice that a RCE `ProxyCertificate` is used for authenticating and no native `GlobusCredential`.


interface
de.rcenvironment.rce.sdk.grid.GridResourceQuerier
queryResources(spec:GridResourceSpecification):List
interface
de.rcenvironment.rce.sdk.grid.GridConnection
submitJob(spec:GridJobSpecification, host:String, port:int):GridJob
de.rcenvironment.rce.grid.globus.GlobusConnection
-myCredentials:GSSCredential=null
-myHost:String=null
-myPort:int=0
+GlobusConnection(host:String, port:int)
+submitJob(specification:GridJobSpecification, host:String, port:int):GridJob
+queryResources(spec:GridResourceSpecification):List
+getGSSCredentials():GSSCredential
+setGGSCredential(cred:GSSCredential):void
de.rcenvironment.rce.sdk.grid.Grid
-Grid()
+authenticate(host:String, port:int, cert:ProxyCertificate):GridConnection
interface
de.rcenvironment.rce.sdk.grid.GridConnector
authenticate(host:String, port:int, cert:ProxyCertificate):GridConnection
de.rcenvironment.rce.grid.globus.GlobusConnector
+HOME_PATH:String=System.getProperty("user.home")
-FILE_SEPARATOR:String=System.getProperty("file.separator")
+authenticate(host:String, port:Int, cert:ProxyCertificate):GridConnection

Name	`GridConnector`
Short description	An interface representing a Grid connector
Extends	
Implements	

Known implementers	Grid, {Globus|Unicore|*}Connector	
Associations		
Attribute	Description	
Method	Description	Parameter
authenticate	Authenticates with a Grid system and creates a certificate for the Grid system by using data available in the RCE ProxyCertificate	**cert** The certificate used to authorize. Returns a GridConnection

Name	*GridResourceQuerier*	
Short description	An interface representing a querier for Grid resources	
Extends		
Implements		
Known implementers	{Globus|Unicore|*}Connection	
Associations		
Attribute	Description	
Method	Description	Parameter
queryResources	Queries the grid system for resources fulfilling the needs specified in the specification.	**spec** A grid resource specification Returns a list of grid nodes

Name	Grid	
Short description	A class representing a connector to the default (configured) Grid System	
Extends		
Implements	GridConnector	
Associations		
Attribute	Description	
Method	Description	Parameter
auth	Authenticates with the underlying grid system and creates a certificate for the grid system by using data available in the RCE ProxyCertificate	**cert** The certificate used to authorize. Returns a GridConnection

Name	GridConnection	
Short description	A class representing an established connection to a Grid System	
Extends		
Implements	GridResourceQuerier	
Known implementers	{Globus|Unicore|*}Connection	
Associations		
Attribute	Description	
Method	Description	Parameter
getHostAddress	Gets the address of the host	
getPort	Gets the port of the host	
submitJob		**spec** The grid job specification Returns a GridJob
queryResources	Queries the grid system for resources	**spec**

	fulfilling the needs specified in the specification.	A grid resource specification Returns a list of grid nodes.

4.2.3 Job Description and Submission

Job description is a very central part of the API that I am designing. The description needs to make it possible to describe everything needed for executing a job, including describing path to the executable, environment variables, resource requirements as well as handing file staging and input and output.

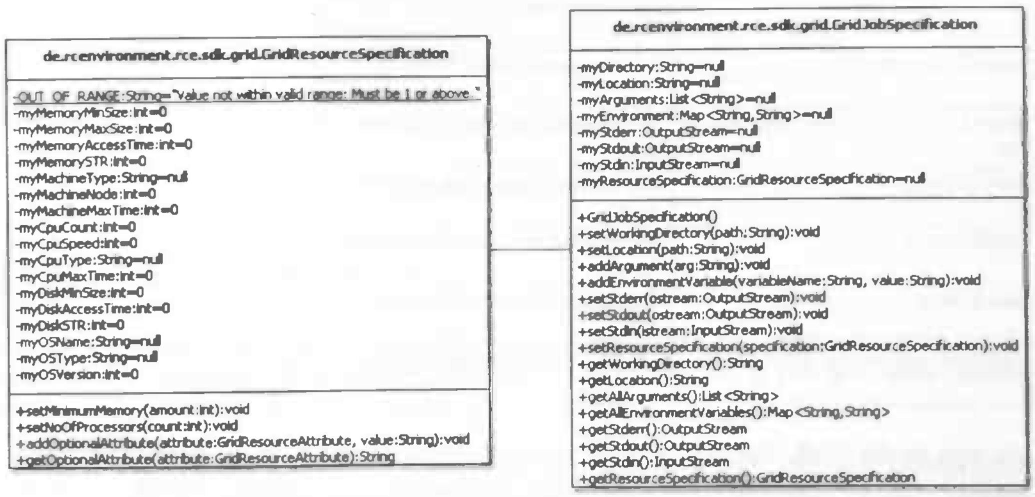
The solution I settled on is to use two description classes. One that represents the job details and another that represents the hardware requirements.

The reason for this is that I can use the hardware description for querying for hardware resources, which mean that the description is not solely job specific.

Since a job is a standard executable it might be possible that it reads and writes output using `stdin`, `stdout` and `stderr`. Most programs write output though, mostly status and error information which can be valuable when things does not work the way they are expected to.

For this reason it will be advantageous handling `stdout` and `stdout` redirection. There are two options here. One is adding API for connecting a Java IO Stream to each of these and the other option is simply putting the output in a log, by using the standard logging system present in RCE. My solution is to log `stderr` and `stdout` in case they are not set by the programmer using my API. Being able to connect a Java `IOStream` is very handy for the user, as the user easily can store the output in a file (by using a `FileOutputStream`) or print directly to the console by using `System.out`.

The design looks like below. The design of the grid resource description is described in the next section.

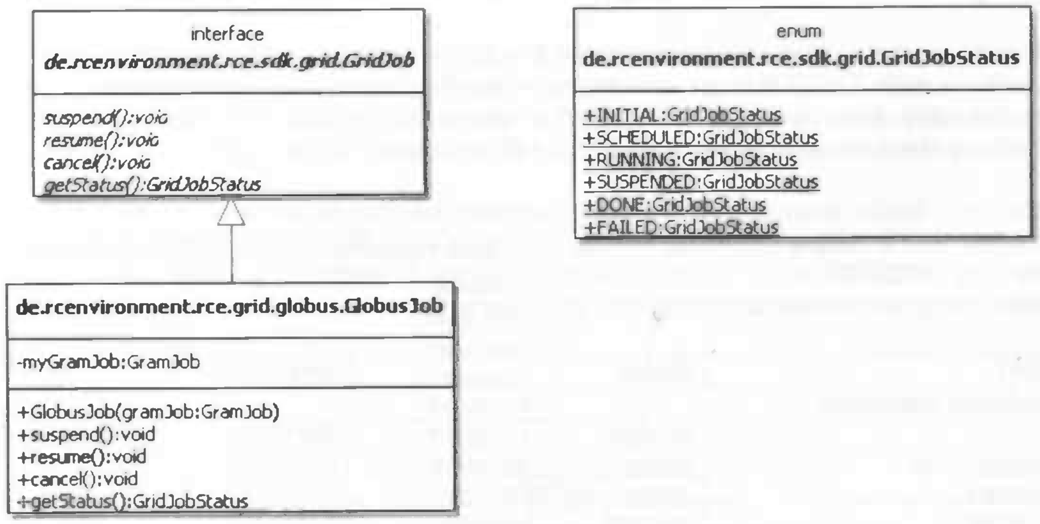

de.rcenvironment.rce.sdk.grid.GridResourceSpecification
-OUT_OF_RANGE:String="Value not within valid range: Must be 1 or above."
-myMemoryMinSize:int=0
-myMemoryMaxSize:int=0
-myMemoryAccessTime:int=0
-myMemorySTR:int=0
-myMachineType:String=null
-myMachineNode:int=0
-myMachineMaxTime:int=0
-myCpuCount:int=0
-myCpuSpeed:int=0
-myCpuType:String=null
-myCpuMaxTime:int=0
-myDiskMinSize:int=0
-myDiskAccessTime:int=0
-myDiskSTR:int=0
-myOSName:String=null
-myOSType:String=null
-myOSVersion:int=0
+setMinimumMemory(amount:int):void
+setNoOfProcessors(count:int):void
+addOptionalAttribute(attribute:GridResourceAttribute, value:String):void
+getOptionalAttribute(attribute:GridResourceAttribute):String
de.rcenvironment.rce.sdk.grid.GridJobSpecification
-myDirectory:String=null
-myLocation:String=null
-myArguments:List<String>=null
-myEnvironment:Map<String, String>=null
-myStderr:OutputStream=null
-myStdout:OutputStream=null
-myStdin:InputStream=null
-myResourceSpecification:GridResourceSpecification=null
+GridJobSpecification()
+setWorkingDirectory(path:String):void
+setLocation(path:String):void
+addArgument(arg:String):void
+addEnvironmentVariable(variableName:String, value:String):void
+setStderr(ostream:OutputStream):void
+setStdout(ostream:OutputStream):void
+setStdin(istream:InputStream):void
+setResourceSpecification(specification:GridResourceSpecification):void
+getWorkingDirectory():String
+getLocation():String
+getAllArguments():List<String>
+getAllEnvironmentVariables():Map<String, String>
+getStderr():OutputStream
+getStdout():OutputStream
+getStdin():InputStream
+getResourceSpecification():GridResourceSpecification

Name	`GridJobSpecification`

Short description	A class representing a job to be executed on the Grid System	
Extends		
Implements		
Associations	`GridResourceSpecification`	
Attribute	**Description**	
`directory`	The default working directory	
`location`	The path (relative to working directory) to the executable, incl. the name of the executable.	
`arguments`	The arguments to be given to the executable.	
`environment`	The environment variables to be set before executing the executable.	
`stderr`	Standard error redirection	
`stdout`	Standard out redirection	
`stdin`	Standard in redirection	
`resourceSpecification`	A resource specification	
Method	**Description**	Parameter
`setWorkingDirectory`	Set the working directory	**path** The working directory
`setLocation`	Set the path to the executable	**path** The executable
`addArgument`	Adds command line arguments that will be given to the executable upon job execution.	**arg** The argument to add.
`addEnvironmentVariable`	Adds environment variables needed by the executable	**variable** The environment variable to set.
		value The value of the variable.
`setStderr`	Sets the output stream to redirect standard err to.	**stream** The corresponding stream
`setStdout`	Sets the output stream to redirect standard out to.	**stream** The corresponding stream
`setStdin`	Sets the Input stream to redirect standard in to.	**stream** The corresponding stream
`setResourceSpecification`	Adds a resource spec.	**spec** The specification
`getWorkingDirectory`	Gets the working directory	
`getLocation`	Gets the path to the executable	
`getAllArguments`	Gets all arguments that has been added	
`getAllEnvironmentVariabl es`	Gets environment variables that have been added	
`getStderr`	Gets the output stream to redirect standard err to.	
`getStdout`	Gets the output stream to redirect standard out to.	
`getStdin`	Gets the input stream to redirect standard in to.	
`GetResourceSpecification`	Gets the associated resource specification.	spec The specification

As seen in the UML diagram of `GridConnection` you submit a job using the `sumbitJob()` method of `GridConnection` and supplying a `GridJobSpecification`. The user needs to be able to suspend, resume and cancel a job as well as get the status. This is important as grid jobs often are batch jobs that run for a long time (for instance months).

The way this has been designed is that the `sumbitJob()` method returns a `GridJob` object which offers this functionality. The design of `GridJob` is shown below.



Name	*GridJob*	
Short description	An interface representing a job to be executed on the Grid System	
Extends		
Known implementers	`{Globus|Unicore|*}Job`	
Associations		
Attribute	**Description**	
`jobSpecification`	An associated job specification.	
Method	**Description**	**Parameters/Notes**
`suspend`	Suspends the job	
`resume`	Resumes a suspended job	
`cancel`	Cancels a submitted job	
`getStatus`	Returns status info	`INITIAL, SCHEDULED, RUNNING, SUSPENDED, FAILED, DONE`

4.2.4 Grid Resource Description

A grid consists of resources such as storage, processing power etc. These resources can be used in two situations; when querying for resources needed for a job and when adding resource requirements to a job execution.

The way resources are handled in UNICORE and Globus is very different [Bihler2005, Brooke, Brook-2]. The latest versions of Globus uses the OSGA approved GLUE[24] schema for resource description. GLUE stands for Grid Laboratory Uniform Environment, and was

24 http://www.hicb.org/glue/glue-v0.1.2.doc

developed by DataTAG[25] in order to insure interoperability between US and Europe Grid domains. GLUE describes resources independently from their implementation architecture and language, where as UNICORE describes resource requests and resources information in a hierarchy of serializable Java classes by using Abstract Job Objects (AJO) to discover and manage resources as well as managing the Grid jobs.

This means that Globus only lets you make few resource requirements when submitting jobs where as with UNICORE you can require everything specified. If you want to ensure requirements with Globus, you first need to query for resources and then select a node fulfilling the requests. Support for doing this will be designed in this section.

The GAT Toolkit described later provides a common interface to resources in UNICORE and Globus, so it is interesting looking at what they make available. The GAT model is similar to the way UNICORE works so let's compare it to what is available in Globus and select the basic things we need and design my API according to this.

GAT	*Globus*	*Where found*
Software Description		
	libraryPath	GRAM WS SPEC [26]
Working Directory	directory	GRAM RSL/CoG[27]
Arguments	argument	GRAM RSL/CoG
Environment	environment	GRAM RSL/CoG
Location	executable	GRAM RSL/CoG
StdErr	stderr	GRAM RSL/CoG
StdOut	stdout	GRAM RSL/CoG
StdIn	stdin	GRAM RSL/CoG
Software Resource Description		
"os.name"	hostOSName	GRAM WS RP[28]
"os.type"	hostOSType	GRAM WS RP
"os.version"	hostOSVersion	GRAM WS RP
"os.release"		
Hardware Resource Description		
"memory.size"	minMemory	GRAM RSL/CoG
"memory.accesstime"		
"memory.str"[29]		
"machine.type"		
"machine.node"		
"cpu.count"	count	GRAM RSL/CoG / GRAM WS RP
"cpu.speed"		
"cpu.type"	hostCPUType	GRAM WS RP
"disk.size"		
"disk.accesstime"		
"disk.str"		
	maxMemory	GRAM RSL/CoG
	maxCpuTime	GRAM RSL/CoG / GRAM WS RP
	maxWallTime	GRAM RSL/CoG / GRAM WS RP
	maxTotalTime	GRAM RSL/CoG / GRAM WS RP

[25] http://datatag.web.cern.ch/datatag/
[26] http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html
[27] http://www-unix.globus.org/cog/distribution/1.2/api/org/globus/gram/GramAttributes.html
[28] http://www.globus.org/toolkit/docs/4.0/rp.html
[29] Sustained transfer rate

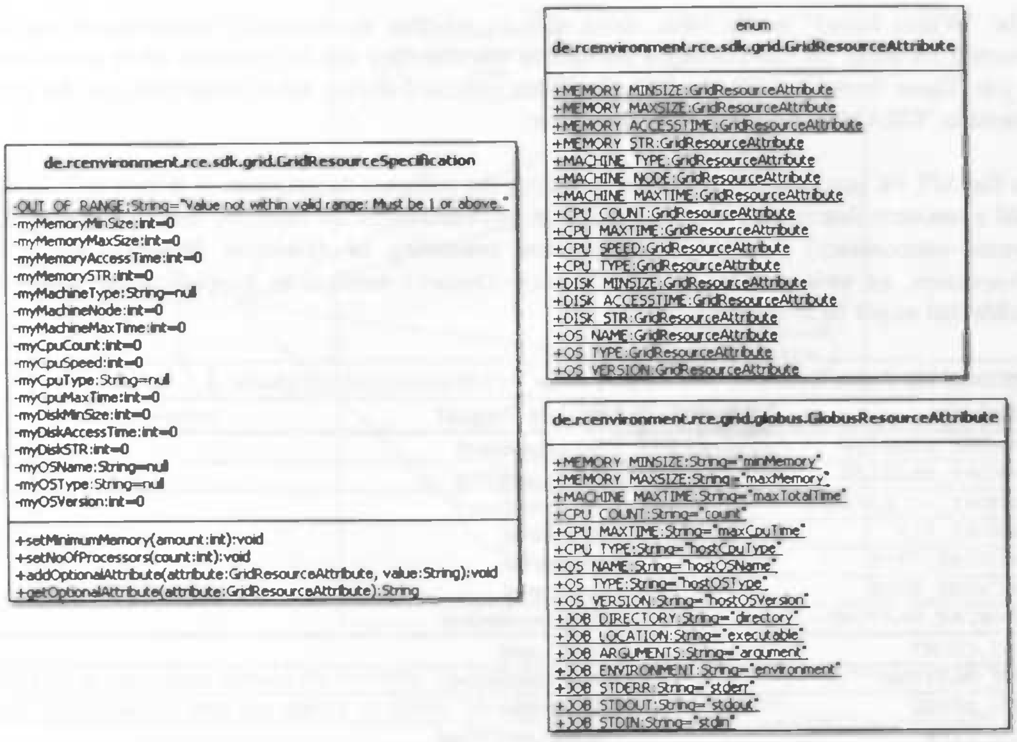
The "Where found" in the table about tells us whether the resource requirements can be queried for using the Globus Index Service or whether they can be enforced when submitting a job. Those found in "GRAM RSL/CoG" are enforced during submission whereas the ones found in "GRAM WS RP" can be queried for.

In the API we need a job description including the software description. It is then possible to add a resource description (the split between software/hardware resource description in GAT seems unnecessary) to this job description consisting of *Minimum Memory* and *# of Processors*, as well as an `addOptionalAttribute()` method to support some specified fields that *might* be available.

Optional Hardware Resource Attributes (`enum GridResourceAttribute` **)**	
Attribute	***Globus Support***
MEMORY_MINSIZE	Always (*minMemory*)
MEMORY_MAXSIZE	Always (*maxMemory*)
MEMORY_ACCESSTIME	Not supported
MEMORY_STR	Not supported
MACHINE_TYPE	Not supported
MACHINE_NODE	Not supported
MACHINE_MAXTIME	Always (*maxTotalTime*)
CPU_COUNT	Always (*count*)
CPU_MAXTIME	Always (*maxCpuTime*)
CPU_SPEED	Not supported
CPU_TYPE	Optional (*hostCPUType*)
DISK_MINSIZE	Not supported
DISK_ACCESSTIME	Not supported
DISK_STR	Not supported
	Always (*maxWallTime*)
OS_NAME	Optional (*hostOSName*)
OS_TYPE	Optional (*hostOSType*)
OS_VERSION	Optional (*hostOSVersion*)

Notes to the above schema: "Optional" means that they are only tested when querying for resources and not with job submission. Testing does not mean that they are available as the fields are optional in the Globus world, and depends whether they have been added to the Resource Discovery Service either manually or by an indexing service.

The design is shown below:


de.rcenvironment.rce.sdk.grid.GridResourceSpecification
-OUT_OF_RANGE:String="Value not within valid range: Must be 1 or above."
-myMemoryMinSize:int=0
-myMemoryMaxSize:int=0
-myMemoryAccessTime:int=0
-myMemorySTR:int=0
-myMachineType:String=null
-myMachineNode:int=0
-myMachineMaxTime:int=0
-myCpuCount:int=0
-myCpuSpeed:int=0
-myCpuType:String=null
-myCpuMaxTime:int=0
-myDiskMinSize:int=0
-myDiskAccessTime:int=0
-myDiskSTR:int=0
-myOSName:String=null
-myOSType:String=null
-myOSVersion:int=0
+setMinimumMemory(amount:int):void
+setNoOfProcessors(count:int):void
+addOptionalAttribute(attribute:GridResourceAttribute, value:String):void
+getOptionalAttribute(attribute:GridResourceAttribute):String
enum
de.rcenvironment.rce.sdk.grid.GridResourceAttribute
+MEMORY_MINSIZE:GridResourceAttribute
+MEMORY_MAXSIZE:GridResourceAttribute
+MEMORY_ACCESSTIME:GridResourceAttribute
+MEMORY_STR:GridResourceAttribute
+MACHINE_TYPE:GridResourceAttribute
+MACHINE_NODE:GridResourceAttribute
+MACHINE_MAXTIME:GridResourceAttribute
+CPU_COUNT:GridResourceAttribute
+CPU_MAXTIME:GridResourceAttribute
+CPU_SPEED:GridResourceAttribute
+CPU_TYPE:GridResourceAttribute
+DISK_MINSIZE:GridResourceAttribute
+DISK_ACCESSTIME:GridResourceAttribute
+DISK_STR:GridResourceAttribute
+OS_NAME:GridResourceAttribute
+OS_TYPE:GridResourceAttribute
+OS_VERSION:GridResourceAttribute
de.rcenvironment.rce.grid.globus.GlobusResourceAttribute
+MEMORY_MINSIZE:String="minMemory"
+MEMORY_MAXSIZE:String="maxMemory"
+MACHINE_MAXTIME:String="maxTotalTime"
+CPU_COUNT:String="count"
+CPU_MAXTIME:String="maxCpuTime"
+CPU_TYPE:String="hostCpuType"
+OS_NAME:String="hostOSName"
+OS_TYPE:String="hostOSType"
+OS_VERSION:String="hostOSVersion"
+JOB_DIRECTORY:String="directory"
+JOB_LOCATION:String="executable"
+JOB_ARGUMENTS:String="argument"
+JOB_ENVIRONMENT:String="environment"
+JOB_STDERR:String="stderr"
+JOB_STDOUT:String="stdout"
+JOB_STDIN:String="stdin"

Name	`GridResourceSpecification`	
Short description	A class representing a resource specification describing resource requirements	
Extends		
Implements		
Associations		
Attribute	**Description**	
`myMemoryMinSize`	Minimum memory to be available (in MB)	
`myMemoryMaxSize`	Maximum memory to be used (in MB)	
`myMemoryAccessTime`	Minimum memory access time	
`myMemorySTR`	Minimum memory sustained transfer rate	
`myMachineType`	Machine type	
`myMachineNode`	Machine note	
`myMachineMaxTime`	Maximum amount of time allowed for job	
`myCpuCount`	Number of processors needed	
`myCpuSpeed`	The clock speed of the CPU	
`myCpuType`	The type of CPU ("i686", "powerpc", etc)	
`myCpuMaxTime`	Maximum CPU time	
`myDiskMinSize`	Minimum disk storage to be available	
`myDskAccessTime`	Maximum disk access time	
`myDiskSTR`	Minimum disk sustained transfer rate	
`myOsName`	The name of the operation system ("linux", "solaris", etc.)	
`myOsType`	The type of operation system	
`myOsVersion`	The version of the operation system	
Method	**Description**	**Parameter**
`setMinimumMemory`	Sets minimum amount of memory required.	**amount** The amount in MB.

setNoOfProcessors	Sets number of processors required.	**count** The no. of processors
addOptionalAttribute	Add optional attribute	**attr** The attribute. Ex.*OS_NAME*.
		value The value. Ex. "linux".
getOptionalAttribute	Gets an optional attribute	

As seen in the UML diagram of `GridConnection` it is possible to use the `GridResourceSpecification` for querying of resources, and as seen in the UML diagram of `GridJobSpecification` it is possible to attach a `GridResourceSpecification` to a given `GridJobSpecification`.

4.2.5 Staging of Data Files

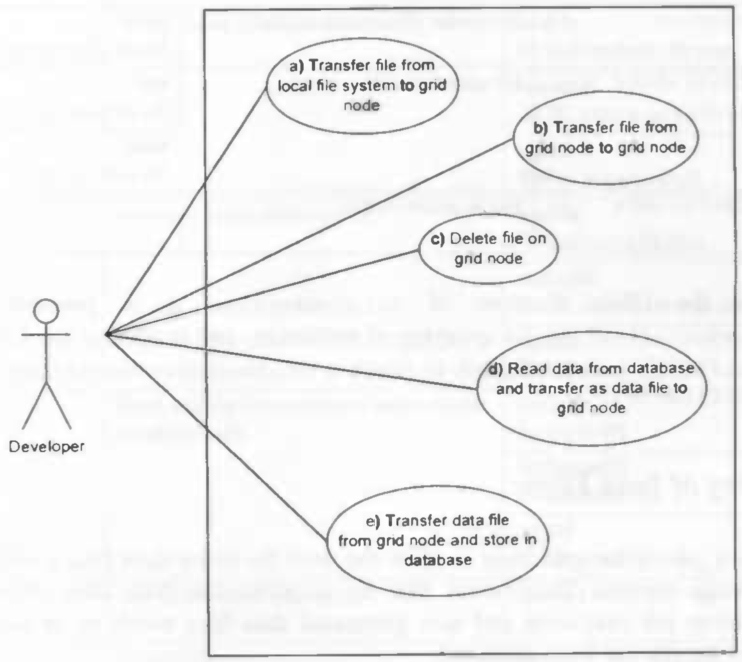
When we run a job on the grid there is often the need for using data files available elsewhere on other storage servers. This means that for a given job data files often needs to be transferred before job execution and new generated data files needs to be stored on remote locations after the job has been executed.

Conceptually, these transfers seem job dependent and it is a good idea to add API for this functionality to the `GridJob` class.

On the other hand these data files might be of reasonable size and might be needed by other jobs as well. In this case it is advantageous not having to transfer the same files again.

Since this is the case, having the API be part of the *Job* class might give the developer a bit of extra headache as he/she will have to know which jobs transferred which files and which jobs has to clean up afterward. Based on this reasoning a separate class will be created to deal with data transfers and clean up, but I will also support for transferring data files along with the jobs – this should give the developer full flexibility.

Let's look at possible use cases for data transfers for job execution:


a) Transfer file from local file system to grid node
b) Transfer file from grid node to grid node
c) Delete file on grid node
Developer
d) Read data from database and transfer as data file to grid node
e) Transfer data file from grid node and store in database

The above use-cases should be covered by simply introducing API with support for node-to-node file transfers plus the ability to delete files from a grid node as well.

a) A user needs to transfer a file from the local file system to a grid node.

   Since the local system has a grid installation the support for node-to-node transfers will take care of this. Additionally, normally the local file system is remotely mounted on the grid node (for instance via NFS) and transfers are not needed at all.

b) A user wants to transfer a file from one grid node to another.

   The node-to-node transfer support again takes care of this.

c) A user wants to delete a file from a grid node as it is no longer needed.

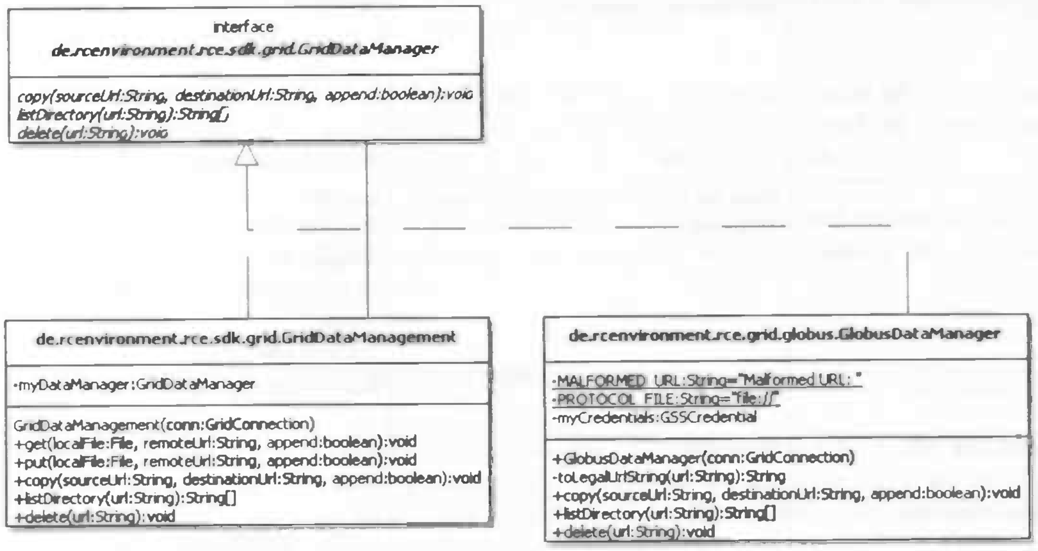
   The delete file support is needed for accomplishing this.

d) A user wants to read data from an RCE database installation and transfer the data as a file to a grid node.

   The Data Management supports requesting data from a RCE database installation and returning it locally as a Java object. This object can then be serialized (for instance as comma separated values) and be accessed via the mounted local file system or transferred via the node-to-node transfer support.

e) A user has executed a job and wants to store the output file in an RCE database installation.

If the local file system is mounted and used by the grid node, the data file will be stored locally; otherwise it can be easily transferred as described above. The file can be read into Java and stored in an RCE database using the currently existing Data Management framework.



interface
de.rcenvironment.rce.sdk.grid.GridDataManager
copy(sourceUrl:String, destinationUrl:String, append:boolean):void
listDirectory(url:String):String[]
delete(url:String):void
de.rcenvironment.rce.sdk.grid.GridDataManagement
-myDataManager:GridDataManager
GridDataManagement(conn:GridConnection)
+get(localFile:File, remoteUrl:String, append:boolean):void
+put(localFile:File, remoteUrl:String, append:boolean):void
+copy(sourceUrl:String, destinationUrl:String, append:boolean):void
+listDirectory(url:String):String[]
+delete(url:String):void
de.rcenvironment.rce.grid.globus.GlobusDataManager
-MALFORMED_URL:String="Malformed URL: "
-PROTOCOL_FILE:String="file://"
-myCredentials:GSSCredential
+GlobusDataManager(conn:GridConnection)
-toLegalUrlString(url:String):String
+copy(sourceUrl:String, destinationUrl:String, append:boolean):void
+listDirectory(url:String):String[]
+delete(url:String):void

Name	*GridDataManager*	
Short description	An interface representing file operations between RCE and Grid systems	
Extends		
Known implementers	GridDataManagement, {Globus|Unicore|*}DataManager	
Associations		
Attribute	Description	
Method	Description	Parameter
copy	Copies data files from remote grid node to another remote grid node.	sourceUrl The source URL
		destinationUrl The destination URL
		append Whether to append or not
listDirectory	Lists the contents of a directory.	Returns a vector of strings
delete	Deletes a file.	url The URL to the file to delete.

Name	GridDataManagement	
Short description	A class representing file operations between RCE and grid systems.	
Extends	GridDataManager	
Associations	GlobusDataManager, UnicoreDataManager	
Attribute	Description	
Method	Description	Parameter

`get`	Makes a remote data file available locally	**localFile** The local destination file name
		remoteUrl The URL where to get the file from.
		append Whether to append or not.
`put`	Makes local data available remotely	**localFile** The local source file name.
		remoteUrl The URL where to store the file.
		append Whether to append or not.
`copy`	Copies data files from remote grid node to another remote grid node.	**sourceUrl** The source URL.
		destinationUrl The destination URL.
		append Whether to append or not.
`listDirectory`	Lists the contents of a directory.	Returns a vector of strings
`delete (1)`	Deletes a file.	**localFile** The local file to delete.
`delete (2)`	Deletes a file.	**url** The URL to the file to delete.

Chapter 5

The chosen implementation

In this chapter I will look at some of the decisions made in the implementation of the designed API that I in the previous chapter described with UML classes.

Most part of the API (except the querying part) has been implemented using an abstraction layer. For this reason I start with looking into the advantages of using such an abstraction layer as well as present the analysis leading to the choice of the CoG toolkit.

Later in the chapter I present the most important implementation details and choices so that it should be easier for a future maintainer understanding and maintaining the current implementation.

5.1 The Use of an Abstraction Layer

Due to the fact that web services introduce an amount of complexity, and due to the fact that the WSRF standard is still under development[30], different organizations have set out to develop native, easy-to-use, Java APIs that wrap the web services. These make it possible to access the grid middleware in a familiar higher level framework.

Since we are using Java there only seems to be advantages for using these. If it turns out that the abstraction API is too limited in an area the concerned code can be written or rewritten using the grid/web services, though I will try to avoid doing this as far as possible and let the developers of the chosen abstraction layer deal with the changes in the upcoming versions of Globus and WSRF. It should be mentioned that it is indeed not possible to implement everything in the design using these abstraction layers and I will use a web service for implementing the querying support.

Let's take a look at our options, the Java Commodity Grid Kit [CoG], The Grid Programming Environment [GPE], and last but not least the Java version of the Grid Application Toolkit [GAT].

5.1.1 The Java Commodity Grid Kit (CoG)

The Commodity Grid [CoG] provides a Java or Python API to the Globus Toolkit which itself is written partly in ANSI C and partly in Java. It thus provides a high-level framework for accessing and working with a Globus Grid, but CoG is much more than just a high-level interface to the C toolkit, and it provides additional functionality such as grid abstractions and providers, workflows support (via Karajan), etc.

The Java version is also used within Globus itself and provides the implementation of the Java-based GSI (Grid Security Infrastructure), the file transfer protocol GridFTP, the

[30] The OASIS standards process has just recently begun.

credential management service myProxy as well as the GRAM (Grid Resource Allocation and Management) service for among other job submission.

Due to the fact that Java CoG is used within Globus and probably will be used more so in the future, it is clear that using CoG will be future proof and stay maintained for years to come. It thus seems as a safe choice and seems to be the "official" preferred way to develop grid applications.

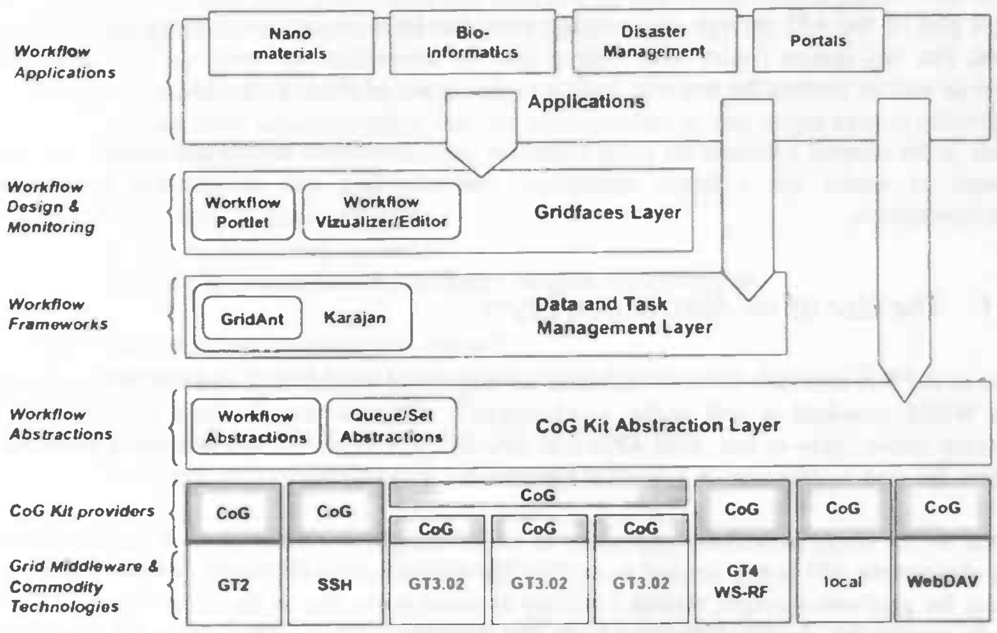


Figure 4: An overview of the CoG architecture from the wiki.[31]

Looking at documentation, CoG is reasonable documented in a wiki (http://wiki.cogkit.org) and there also exist good JavaDoc documentation. The API also seems well thought-out and it is extensible, so that users can include their own abstractions and enhance the functionality of the Java CoG Kit.

The only disadvantage is that it is highly tied to the Globus world and making a UNICODE provider is not feasible according to research[32] done by Intel (see next subsection).

It should be mentioned that though Java CoG is used within Globus, only the needed subset is distributed with Globus and I might need to use the full package instead.

[31] GT is short for the Globus Toolkit
[32] This is stated in the presentations found in the reference section in the appendix under [GPE].

5.1.2 The Grid Programming Environment (GPE)

There is not much information available online about the Grid Programming Environment, but I have been able to get my hands on 3 presentations done by Intel (look at the links in [GPE].) that explain a bit better what it is all about.

From the outside it looks like GPE is just another grid abstraction layer that among others provides support for Globus and UNICORE. GPE is Intel's contribution to the EU sponsored Grid integration project, UniGridS (Uniform Interface to Grid Services) which overall goal is to develop an OGSA compliant Grid Service infrastructure based on UNICORE. Intel is responsible for the "interoperability" part of this project and this is where GPE fits in.

According to Intel GPE (see presentation UniGridS and GPE [GPE])

- Enables applications to run on and across different grid infrastructures incl. UNICORE and Globus.
- Provides a client framework to give users access to the infrastructure.
- Provides the Grid Bean concept and a programming API for Grid developers.
- Will support future virtualization and management concepts.
- Is available under the BSD license.

GPE provides a high-level grid API with descriptions for resources (CIM), jobs (JSDL) and workflows (BPEL) as well as support for various operations within the fields of job management, file transfers, brokering and steering. Additionally, GPE provides a client framework, a grid SDK, as well as grid beans which are grid services combined with a client plug-in.

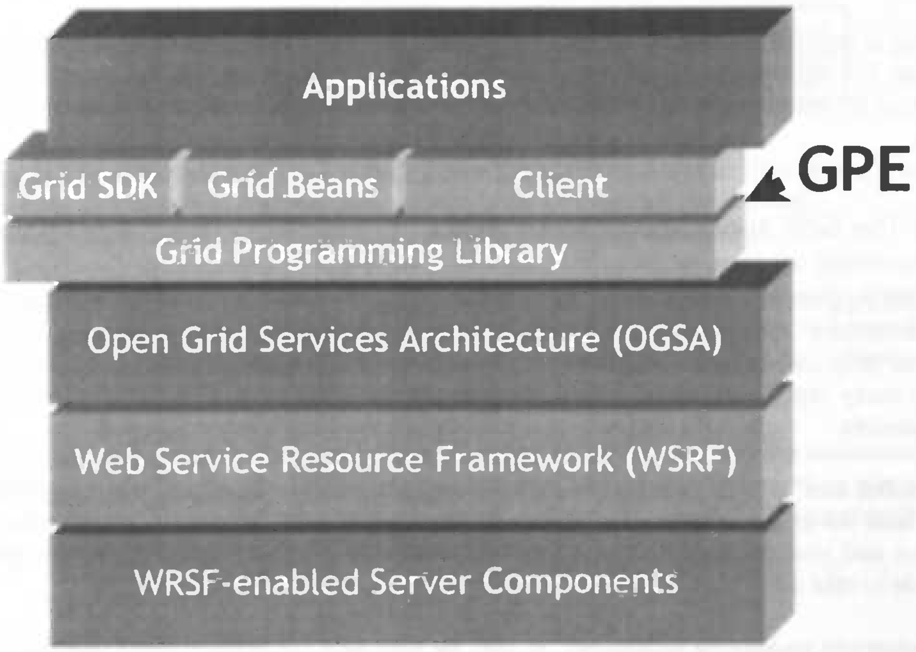

Applications
Grid SDK
Grid Beans
Client
GPE
Grid Programming Library
Open Grid Services Architecture (OGSA)
Web Service Resource Framework (WSRF)
WRSF-enabled Server Components

Figure 4: The architecture of the Intel Grid Programming Environment

From the slides it seems that the design of GPE has been remade between 2004 and end 2005. The slides from 2004 shows that Intel tried writing a UNICORE backend for the Globus CoG toolkit and later integrating UNICORE with the CoG workflow engine Karajan. Intel have then implemented a proof-of-concept implementation based on these ideas and have come to the conclusion that the UNICORE security model was too strict and that the Karajan-XML job description format was too simple (no resource description etc).

Due to these facts Intel have abandoned this idea and instead chosen to cooperate with other standard bodies and implement the following standards:

JSDL (Job Submission Description Language)
- A high level job description that can be submitted to any target system offering a JSDL interface

CIM (Common Information Model)
- A way of describing resources
- Use of CIM management interfaces for grid administration

BPEL (Business Process Execution Model)
- Integration of Grid Bean services into larger business process workflows

Web Service Standards (WS-Addressing, WSRF, WSN, etc)
- Interoperation with other grid middleware

OGSA (Open Grid Services Architecture)
- Share components with other architectures.

Looking at the slides it can be concluded that GPE still has a long way to go to be usable for real use. For this reason I have decided not use GPE at this point. On the other hand it is important to follow where GPE is going to not make our grid support obsolete in the near future.

5.1.3 The Grid Application Toolkit (GAT)

The Grid Application Toolkit is an EU-sponsored project to develop an application toolkit for grid computing. According to one of the developers [GAT], GAT was developed due to the fact that programmers are only slowly accepting the grid computing paradigm and that there are so many rapidly changing grid systems available. Whether this is actually the case, is questionable.

The toolkit can be seen as a simple API for making grid-based or grid-aware applications. GAT uses so-called adaptors for implementing support for the different underlying grid systems and you are always certain that some basic functionality is available, though it is possible to take advantage of platform specific features, on the cost of portability.

GAT abstracts underlying technology, so that the user does not need to know whether a copy is being performed with SSH, FTP, GridFTP, HTTP etc. This is an advantage as the some of these services might not be available on all sites, or might not be available to all users do to restrictions. It also leaves a burden from the programmer and it makes sure that the most suitable underlying technology is used, such as using file-to-file copy locally instead of using a web service to contact another for performing a similar action.

From the looks of it, GAT seems like the heaven for grid programmers; it provides a nice high-level API that is more abstracted than CoG and platform independent, the Java API is even better documented than that of CoG, it is free (BSD license), and it has everything I need.

Looking at the future perspective, GAT provides a stable API that will stay supported for at least the next two years. In the case we need support for an additionally grid system, GAT gives us the option of writing an adaptor for this system. GAT is also being standardized within the Global Grid Forum and the standardized version will be renamed to SAGA (Simple API for Grid Application) and a migration path will be supplied, for easy migration.

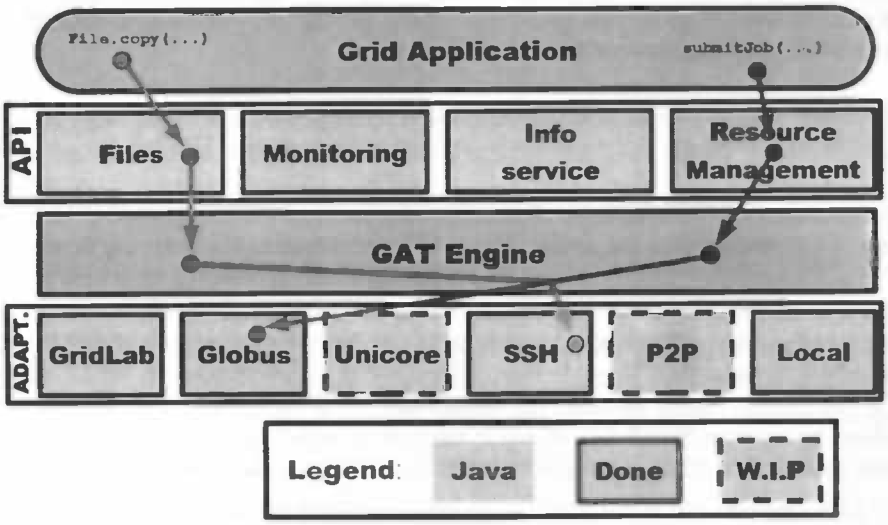


Figure 6: The architecture of GAT showing how the engine chooses the best fitting adaptor.
Copyright, Rob van Nieuwpoort, Vrije Universiteit Amsterdam

There are some disadvantages of using GAT, though. First of all, most of the functionality is not needed for our grid application use-cases as we know our uses and our environment very well. This means that we can select which protocols we want to use and that will provide best performance. One of the biggest disadvantages, though, is that the UNICORE adaptor is not yet fully implemented and tested.

GAT is also a big system with the C-engine taking up 11 MB zipped and the additional Java-API taking up as much as 16.47 MB. It is possible to reduce the size by removing non-needed adaptors but still GAT seems a bit overblown for our limited use, especially considering that we might not be able to use it for UNICORE support.

5.1.4 Conclusion

Having looked at the above "abstractions kits" I have come to the conclusion that the Java CoG kit fits our needs the best. That a limited version of it is being used in Globus itself serves at a stamp of quality and makes us comfortable that CoG will be around and maintained for years to come.

The other solutions are interesting as well and should be reevaluated when we will actually implement the UNICORE in RCE.

GAT provides a nice abstract application API, but the size of the package and our limited needs have made me reject this solution. I am also concerned with the usefulness of GAT in the future, especially how it will keep up with GPE and if it will incorporate the newest features of the grid systems on the market.

GPE is a very interesting project as it does not only try to provide an API for grid application developers in spirit with GAT, but also tries changing the underlying grid systems to be more compatible. It does so by cooperation with standard bodies, specifying new standards based on their experience and prototype implementations, as well as cooperating with the Globus team and actually making changes to UNICORE itself.

5.2 *Web Services*

Though the chosen CoG kit includes abstractions for most of the things needed to implement the design, I will still have to use a web service for implementing the querying support.

A web service is not to be confused with a website of any kind. Basically a web service is a software system developed to allow interoperable machine-to-machine communication over a network (often the internet). The interface of the system is described by a WSDL description which is an XML based format for describing these services. The WSDL standard standardizes how to use SOAP for talking to the system which is an extensible framework for packaging and sending XML messages between a service provider and requester.

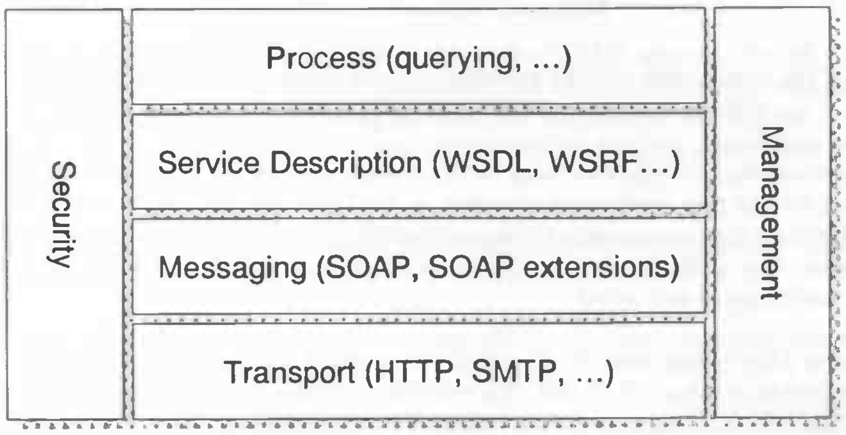

Security
Process (querying, ...)
Service Description (WSDL, WSRF...)
Messaging (SOAP, SOAP extensions)
Transport (HTTP, SMTP, ...)
Management

Figure: An overview of the specifications defining the web service architecture.

Commonly SOAP is transferred over the HTTP transport protocol, though that doesn't have to be the case. Put simple, it can be said that web services just consists of standardized ways of exchanging XML. Since understanding and manipulating XML can be tedious, there exist compilers for most modern programming languages that can generate proxy code that automatically maps the XML messages into native language constructs.

A good source of information on developing WSRF web services together with Globus is the "The Globus Toolkit 4 Programmer's Tutorial" [ProgTut]. The sample code distributed with the tutorial contains the tools necessary for generating Java stubs for the written web service.

5.3 Implementation Details

Implementation details are as the word says, often "details" and are as such less interesting to a thesis. On the other hand, it is important to talk about the design decisions so that another developer can extend and maintain the project after the thesis period. Having talked about grid technology and mentioned that grid systems are based on services it is also interesting knowing which services that has been used in order to implement the Globus part of the final design.

Implementing the described API requires extensions to RCE in two places; the SDK where the public methods and interfaces will be implemented and direct in RCE where the middleware specific implementations will be.

5.3.1 Pitfalls and Problems encountered

When implementing the designed API, I have gained quite a bit of experience and I have also made some mistakes along the way. Whereas Globus is more-or-less well documented, CoG is not. For instance, more of the examples found on the wiki-page (http://wiki.cogkit.org) simply don't work.

The biggest problem I have had is that I ended up implementing the designed API twice, as CoG has two different API's for working with Globus. From reading the overview of CoG, it was our understanding that CoG was an abstraction layer for the Globus web services that is more integrated with Java as well as many extensions such as workflow management, and the like. CoG was also promoted as being of high quality due to the fact that parts of it (some of the Java JAR files) are included with Globus.

Looking at the examples found on the internet and the Javadoc of the JAR files, revealed two API's. One a bit more low-level that the other and the other a bit more complex and integrated with various other (for us) non-relevant CoG services. Due to the fact that the first was distributed with Globus and that most examples used this API, I chose to base the implementation on this.

Unfortunately, many things didn't work and it turned out that the reason was that the API talked to the old C-based grid services and not the new web serviced based ones. The C-based grid services are distributed with Globus, but are deprecated and are not configured on our test system. Founding out that I had used the wrong API came as a shock so a future maintainer should be aware of the differences of the two API's.

Another problem is the JAR files. CoG and Globus consists of many JAR files (code library packets) and from reading the Javadoc it is not possible to see in which JAR file the classes are located. For this reason I wrote a simple Perl script that listed all files in each JAR file, so that it was easy to search for a class and finding the responding JAR file.

Unfortunately, a lot of dynamical class loading is done in CoG (and to some extend in Globus as well), which means that if the code compiled with a number of JAR files, it might still not run before adding more JAR files to your `$CLASSPATH`. Often you will get a `ClassNotFoundException` at runtime, but at other times things just doesn't work. For instance you cannot make local file transfers before you add `cog-provider-local.jar` to your class path.

There have also been some similar problems when writing the querying support that is implemented using a web service API. These problems will be described in the section about querying.

It should also be mentioned that you easily can get in the situation that something doesn't work due to the fact that the grid services used have not been properly configured with Globus. Though, it is not always a simple job configuring these, Globus come with a lot of documentation at the Globus website to help you along, as well as mailing lists where you can communicate and ask the developers and other users more specific questions.

In the following subsections I will look at the different parts of the API and talk about how it has been implemented. This should help a future maintainer to understand the code.

5.3.2 Job Description and Submission

Job submission is handled by the GRAM service[33]. GRAM, which is an abbreviation of Grid Resource Allocation and Management, provides a single interface for requesting and using remote system resources for the execution of grid jobs. It is designed to provide a uniform, flexible interface to job scheduling systems.

CoG provides an API that makes it possible to talk to the GRAM service. The job submission part of the design is implemented by generating a job and resource description that the GRAM service understands and then simply taking advantage of the features that GRAM has to offer offers. How the files are staged and how input and output are handled is described in the next two sections.

GRAM uses the XML format for describing jobs and the resources they require. CoG supplies convenience functions so that you do not need to deal with the XML.

Below I show parts of the code used to convert out `GridJobDescription` into the XML format using the CoG convenience functions.

```
JobSpecification attributes = new JobSpecificationImpl();

// Standard job attributes

String directory = specification.getWorkingDirectory();

if (directory != null) {
```

[33] GRAM Service description; http://www.globus.org/toolkit/docs/4.0/execution/wsgram/

```
    attributes.setDirectory(directory);
}

String location = specification.getLocation();

if (location != null) {
     attributes.setExecutable(location);
}

// Add arguments.
for (String arg: specification.getAllArguments()) {
    attributes.addArgument(arg);
}

// Add environment variable.
Map<String, String> environmentVariables =
   specification.getAllEnvironmentVariables();
Iterator keyValuePairs = environmentVariables.entrySet().iterator();
for (int i = 0; i < environmentVariables.size(); i++) {
    Map.Entry entry = (Map.Entry) keyValuePairs.next();
    String variable = (String) entry.getKey();
    String value = (String) entry.getValue();

    attributes.addEnvironmentVariable(variable, value);
}

...
```

Since it is possible to enforce some of the hardware resource requirements I do this as well:

```
   Globus supported and job hornored hardware resource attributes
GridResourceSpecification resourceSpecification =
    specification.getResourceSpecification();

if (resourceSpecification != null) {

   String memMinSizeValue = resourceSpecification
       .getOptionalAttribute(GridResourceAttribute.MEMORY_MINSIZE);

   if (memMinSizeValue != null) {
       attributes.setAttribute(GlobusResourceAttribute.MEMORY_MINSIZE,
                               memMinSizeValue);
   }

   ... and so on
}
```

When everything has been set up such as input and output handing, hardware resource requirements, etc it is possible using the GRAM service for submitting the job, all that is required is the XML description and valid credentials. The code for doing this is similar to the following code snippet, though the real implementation deals with error handling.

First I set up the CoG provider (In our case “gt4” – Globus 4), the service contract and the service contact (host name plus port).

```
String provider = "gt4";
String jobmanager = "fork";
String serviceContact = host + ":" + port;
```

Now I create a `GlobusJob` (which is also a `GridJob`) object so that I can pass along the needed instantiated classes so that it is possible for the user to later cancel, pause and resume the execution of the job using this `GridJob` object.

I also create a job submission task, sets the CoG provider and the job manager, and instantiate the security context (with our Globus credentials). I also set up a status listener that can be used to perform tasks then the job has finishes, such as staging out files.

```
GlobusJob job = null;
task = new TaskImpl("myTask", Task.JOB_SUBMISSION);

// Submit the job

ExecutionService service = new ExecutionServiceImpl();
service.setProvider(provider);

SecurityContext securityContext = null;
securityContext = AbstractionFactory.newSecurityContext(provider);
securityContext.setCredentials(myCredentials);
service.setSecurityContext(securityContext);

ServiceContact sc = new ServiceContactImpl(serviceContact);
service.setServiceContact(sc);

service.setJobManager(jobmanager);

task.addService(service);

TaskHandler handler = null;

handler = AbstractionFactory.newExecutionTaskHandler(provider);

task.setSpecification(attributes); // add the job description

job = new GlobusJob(handler, task);
job.setStdoutName(stdoutName);
job.setStderrName(stderrName);
job.setSpecification(specification);
...

task.addStatusListener(new GlobusJobStatusListener(job, host, port, this));

handler.submit(task);

return job;
```

5.3.3 Handling of Input and Output

Handling of input and output redirection is supposed to be done by starting and using a local GASS server. GASS, which stands for Global Access to Secondary Storage, is a service that makes it possible to transfer files from grid nodes to grid nodes using the HTTP protocol used by web browsers for fetching web pages. By using GASS I stream output from the jobs over the HTTP protocol and directly into Java IOStream streams.

Here I show the code for handing the output streams with the use of GRAM

```
if (specification.getStderr() != null || specification.getStdout() != null)
{
    // Start a GASS server on client machine.
    GassServer gass = null;

    int options = GassServer.READ_ENABLE | GassServer.WRITE_ENABLE |
        GassServer.STDOUT_ENABLE | GassServer.STDERR_ENABLE;

    try {
        // setting the port to 0 means that a port is assigned dynamically.
        gass = new GassServer (myCredentials, 0);
        gass.registerDefaultDeactivator();
        gass.setOptions(options);
    } catch (IOException e) {
        throw new IOException("Redirection failure: " + e.getMessage());
    }

    if (specification.getStdout() != null) {
        attributes.add(GlobusResourceAttribute.JOB_STDOUT, gass.getURL() +
            "/dev/stdout");
        gass.registerJobOutputStream("out", specification.getStdout());
    }

    if (specification.getStderr() != null) {
        attributes.add(GlobusResourceAttribute.JOB_STDERR, gass.getURL() +
            "/dev/stderr");
        gass.registerJobOutputStream("err", specification.getStderr());
    }
}
```

And the listener which uses a `StringBuffer` for storing the output data:

```
public class GlobusOutputListener implements JobOutputListener {

    private StringBuffer myJobOutput = new StringBuffer("");

    public void outputChanged(String output) {
        myJobOutput.append(output);
    }

    public void outputClosed() {
        // Do nothing
    };

    public String getOutput() {
        return myJobOutput.toString();
    }
}
```

Unfortunately the above didn't work as advertised due to the fact that the Globus 4 CoG provider doesn't yet support this. The above code is left in the implementation as it should work with older providers and should work in the future as well.

As the above elegant solution didn't work I implemented another way of handing the above. The idea is writing the output in a file on the remove grid node and then when the job has finished transfer this output file to the local machine and convert it into a Java `OutputStream`.

Snippets of the code for doing this are shown below:

```
if (provider.startsWith("gt4") || provider.startsWith("GT4")) {
    Long num = task.getIdentity().getValue();
    stdoutName = TEMP_DIR + DIR_SEPARATOR + "stdout-remote-" + num;

    attributes.setStdOutput(stdoutName);
}
```

The following snippet shows part of the code in the `GlobusStatusListener` (with error handling not shown). The callback function is called when the job status changes, and if the job is completed I transfer the output file and convert it into the output stream set by the user as part of the job description.

In the actual implementation the copying code is factored out in a separate function as I also transfer the output data when the job is suspended. A bit of extra work is needed as I have to clear the output file, so that next time I transfer output I won't transfer the same output once again.

```
public class GlobusJobStatusListener implements StatusListener {

    ...

    public void statusChanged(StatusEvent event) {
        Status status = event.getStatus();

        if (status.getStatusCode() == Status.COMPLETED) {

            ...

            if (myJob.getStdoutName() != null) {

                String remote = "gsiftp://" + myHost +
                    myJob.getStdoutName();
                String local = "gsiftp://127.0.0.1" +
                    myJob.getStdoutName() + "-transferred";

                manager.copy(remote, local, false);

                File outputFile = new File(myJob.getStdoutName());
                BufferedReader br;

                OutputStream outputStream = spec.getStdout();
                BufferedWriter bw;

                br = new BufferedReader(new FileReader(outputFile));
                bw = new BufferedWriter(new OutputStreamWriter
                    (outputStream));

                String line;

                while (true) {
                    if ((line = br.readLine()) == null) {
                        break;
                    }

                    bw.write(line);
                }
                bw.write("\n");
                bw.flush();
```

```
                    }
                }
            }
        }
}
```

Unfortunately, even though the job has the completed status, the files written by the job (output files, stdout, stderr etc) are not necessarily closed and can thus not be transferred. A way to work around this, albeit a bit hacky, is to insert a wait before the transferring the files.

```
Thread.sleep(5000);
```

5.3.4 Staging of Data Files

For staging data files, I have designed an API that makes it possible to transfer files from grid node to grid node, plus made it possible to stage data along with jobs.

Implementing the file staging for job submission is supposedly quite easy as Globus support it out of the box; where as the non-job dependent file transfer support is a bit more complicated.

There are various services that can be used for accomplishing this file transfer support, such as the above mentioned GASS service or the more advanced and performance favorable GridFTP service.

The way this has been implemented the URL defines which service to use. For instance, if the URL starts with "`gsiftp://`", the Globus GridFTP service is used.

It should be mentioned that there exists a new service in the newer Globus releases that is called Reliable File Transfer (RFT) service. RFT is a web service that provides "job scheduler"-like functionality for data movement.

You simply provide a list of source and destination URLs (including directories or file globs) and then the service writes your job description into a database and then moves the files on your behalf. Once the service has taken your job request, interactions with it are similar to any job scheduler.

This service has a few advantages over GridFTP, which is in fact not a web service, in that it doesn't require the client to maintain an open socket connection to the server throughout the whole transfer, which can be inconvenient for longer data transfers. Especially in the case of a failure of the client or the client's host, a recovery is not possible with GridFTP as the information needed for recovery is held in the memory of the client. This is possible by using RFT.

Due to the fact that RFT is relatively new and haven't seen as must testing and usage as GridFTP it is decided to stick with GridFTP, but adding support for RFT is definitely an option for future work.

Working with GridFTP through CoG is quite straight forward, and CoG even provides an abstraction that supports "`file://`", and "`http://`" URLs as well.

Unfortunately, a bit of extra work needs to be done to insure that the URLs generated by Java, for instance from File objects can be understood by Globus.

```
private String toLegalUrlString(String url){
    String urlPath;

    // Not file URL. "file:/" only works locally
    if (!url.startsWith("file:")){
        return url;
    }

    // Illegal (in Globus sense) URL (ex. Windows URL generated by Java)
    if (!url.startsWith("file://")){
        urlPath = url.substring(6, url.length());
    } else {
        urlPath = url.substring(7, url.length());
    }

    return "file://127.0.0.1" + urlPath;
}
```

The following simplified code snippet shows how a file can be transferred with the use of CoG. The code is somewhat similar to the code for submitting a job to the GRAM service; i.e. you create a task, setup up the security context, set the "specification" and submit the task.

```
Task task = new FileTransferTask("myTransfer");

URI source = new URI(toLegalUrlString(sourceUrl));
URI destination = new URI(toLegalUrlString(destinationUrl));

FileTransferSpecification spec = new FileTransferSpecificationImpl();
spec.setSource(source.getPath());
spec.setDestination(destination.getPath());
task.setSpecification(spec);

   // SOURCE

SecurityContext sourceSecurityContext = null;
sourceSecurityContext = AbstractionFactory
    .newSecurityContext(source.getScheme());

sourceSecurityContext.setCredentials(myCredentials);

ServiceContact sourceServiceContact = new ServiceContactImpl();
sourceServiceContact.setHost(source.getHost());
sourceServiceContact.setPort(source.getPort());

Service sourceService = new ServiceImpl(
     source.getScheme(), Service.FILE_TRANSFER,
     sourceServiceContact, sourceSecurityContext);

   // TARGET

SecurityContext destinationSecurityContext = null;
destinationSecurityContext = AbstractionFactory
     .newSecurityContext(destination.getScheme());

destinationSecurityContext.setCredentials(myCredentials);
```

```
ServiceContact destinationServiceContact = new ServiceContactImpl();
destinationServiceContact.setHost(destination.getHost());
destinationServiceContact.setPort(destination.getPort());

Service destinationService = new ServiceImpl(
    destination.getScheme(), Service.FILE_TRANSFER,
    destinationServiceContact, destinationSecurityContext);

// add the source service at index 0
task.setService(Service.FILE_TRANSFER_SOURCE_SERVICE, sourceService);

// add the destination service at index 1
task.setService(Service.FILE_TRANSFER_DESTINATION_SERVICE,
    destinationService);

// PERFORM

TaskHandler handler = new GenericTaskHandler();
handler.submit(task);
```

The above code (with error handling etc) has been used to implement the Globus implementation of the `copy()` function of the `DataManagement` API. The other copy functions such as `get()` and `put()` have been implemented using this function.

Staging of input and output was first implemented by using the build-in job staging; meaning that you add the files to the job description. For this to work, it requires a working RFT service, but even with this service working, I couldn't get the build-in staging to work.

As a result of this the input and output staging was reimplemented by using the copy function of the `DataManagement` to copy the input files before job submission and to copy the output files back to the client after the job has finished. This required the following code in `submitJob()` in the class `GlobusConnection` and similar code in `statusChanged()` in the `GlobusJobStatusListener` class:

```
// Stage in files

Map<String, String> stageInUrls = specification.getAllStageInFiles();
Iterator stageInValuePairs = stageInUrls.entrySet().iterator();
for (int i = 0; i < stageInUrls.size(); i++) {
     Map.Entry entry = (Map.Entry) stageInValuePairs.next();
     String local = (String) entry.getKey();
     String remote = (String) entry.getValue();

     if (!local.contains("://"))
     {
         local = "gsiftp://127.0.0.1" + local;
     }

     if (!remote.contains("://"))
     {
         remote = "gsiftp://" + host + remote;
     }

     manager.copy(local, remote, false);
}
```

5.3.5 Job Handling and Status

Implementing support for suspending, resuming and cancelling jobs is as simple as calling one or a few Globus methods:

```
public void suspend() throws SystemException, UserException {
    try {
        myHandler.suspend(myTask);
    } catch (InvalidSecurityContextException e) {
        throw new UserException(e);
    } catch (TaskSubmissionException e) {
        throw new SystemException(e);
    }
}
```

On the other hand the job status from Globus cannot be used without modification. First of all the different status messages have to be mapped to the status messaged used in our API, but since we transfer output and output files (staging out) when the job has been completed, we cannot change the status to "DONE", before we are sure that this data transfer has been completed:

```
public GridJobStatus getStatus() {
    int gstatus = myTask.getStatus().getStatusCode();

    GridJobStatus status = GridJobStatus.INITIAL;

    switch (gstatus){
    case Status.SUBMITTED:
        status = GridJobStatus.SCHEDULED;
        break;
    case Status.ACTIVE:
        status = GridJobStatus.RUNNING;
        break;
    case Status.COMPLETED:
        if (isFinished){   ◀ Make sure we've really finished
            status = GridJobStatus.DONE;
        } else {
            status = GridJobStatus.RUNNING;
        }
        break;

      ... (several cases have been removed from the example)

    default:
        status = GridJobStatus.UNKNOWN;
    }

    return status;
}
```

5.3.6 Querying Support

As stated earlier I will need to talk to a web service in order to implement querying support. Globus offers a set of information services commonly referred to as MDS, which is short for *Monitoring and Discovery System.*

MDS consists of a set of web services that can be used to monitor and discover services and resources on grid systems. In order to implement querying support I only need to concentrate on one of these, the so-called *Index Service*, which is a WSRF service for querying resource property information.

The service is somewhat similar to UDDI[34], but offers more flexibility. The way it works is that indexes collect various information and publishes it as resource properties which then can be queried using a standard WSRF resource property query. The indexes have a lifespan and if an entry is not refreshed before the lifespan runs out it is removed from the index.

In order to use MDS, it needs to be running in a Globus container on the grid, and a one of the two products, Ganglia or Hawkeye needs to be installed. Out of the box, MDS includes an information provider for these two systems.

The Ganglia information provider gathers cluster data from any resource running Ganglia. IT does so using an XML mapping of the earlier mentioned GLUE schema and then published the information to the GRAM service. MDS then support querying the GRAM service. Hawkeye works similarly, but gathers data about Condor[35] pool resources.

These information providers provide information about the host, memory size, processor load etc, and thus fulfill our needs.

In order to talk to the MDS service, I need to talk though the MDS web service. This is not done by manually sending SOAP messages directly, but instead Java stubs are generated using the Globus "`globus-build-service.sh`" tool. Using these stubs the web service will be accessed as was it written in Java.

What is very interesting is that the MDS querying is not implemented as separate MDS service. Actually, if you look at the API of the MDS services you will notice that there is no API supporting querying MDS. Instead all the index data of MDS are "published" as standard WSRF resource properties, which are normally used for holding state information and making WSRF web services stateful. These resources properties are stored as an XML document, referred to as the Resource Properties Document.

This means that for querying MDS you need to contact the `WSResourcePropertiesService` which is a part of the WSRF standard and then you make querying using an XML querying language.

The central code needed for making queries against the web service is shown below.

```
import java.util.List;

import
    org.oasis.wsrf.properties.WSResourcePropertiesServiceAddressingLocator;
import org.oasis.wsrf.properties.QueryResourceProperties_Element;
```

[34] Universal Description, Discovery and Integration; http://en.wikipedia.org/wiki/UDDI

[35] Condor is a specialized workload management system for compute-intensive jobs and not really a grid middleware system, but it incorporates many of the grid methodologies and protocols and is fully interoperable with resources managed by Globus.

```
import org.oasis.wsrf.properties.QueryResourcePropertiesResponse;
import org.oasis.wsrf.properties.QueryExpressionType;
import org.oasis.wsrf.properties.QueryResourceProperties_PortType;

import de.rcenvironment.rce.sdk.exception.SystemException;

import org.globus.wsrf.WSRFConstants;
import org.globus.wsrf.utils.AnyHelper;
import org.globus.wsrf.utils.FaultHelper;

package de.rcenvironment.rce.grid.globus;

public class QuerySupport {

    static {
        Util.registerTransport();
    }

    public static String queryResources(String expression)
         throws SystemException {

        String dialect = WSRFConstants.XPATH_1_DIALECT;

        WSResourcePropertiesServiceAddressingLocator locator =
            new WSResourcePropertiesServiceAddressingLocator();

        try {
            QueryExpressionType query = new QueryExpressionType();
            query.setDialect(dialect);
            query.setValue(expression);

            QueryResourceProperties_PortType port =
                locator.getQueryResourcePropertiesPort(client.getEPR());

            QueryResourceProperties_Element request
                = new QueryResourceProperties_Element();

            request.setQueryExpression(query);

            QueryResourcePropertiesResponse response =
                port.queryResourceProperties(request);

            return AnyHelper.toSingleString(response);

        } catch (Exception e) {
                throw new SystemException("Error: " +
                   FaultHelper.getMessage(e));
        }
    }
}
```

Notice that the expressions have to be written using XPath[36], which is the official language for addressing parts of an XML document. The actual implementation converts the `GridResourceSpecification` into XPaths and uses these for doing the actual querying.

An example of an XPath is:

[36] XML Path Language (XPath); http://www.w3.org/TR/xpath

```
string(//*[local-name()='GLUECE']/glue:ComputingElement/glue:Info/@glue:TotalCPUs)
```

The above XPath will return the TotalCPUs attribute or an XML `<Info>` tag embedded in a `<ComputingElement>` tag.

Since I need to query for many properties in order to check if our resource requirement description is fulfilled, it is advantageous just querying with "`//*`" as this will return the full Resource Property (XML) Document. The reason for this is that the XPath queries though the web service are slow, and parsing the XML document locally is many times quicker. Actually, in the implementation XPaths are also used for parsing locally by the use of the dom4j XML parser library as the following code sample shows:

```
Document document = DocumentHelper.parseText(resourcePropertyDocument);

Node node;
String value;
String attribute;

value = spec.getOptionalAttribute(GridResourceAttribute.CPU_COUNT);

if (value != null) {
    node = document.selectSingleNode(
        "//*[local-name()='GLUECE']/glue:ComputingElement/glue:Info");

    attribute = node.valueOf("@glue:TotalCPUs");

    if (!value.equals(attribute) && !attribute.equals(NOT_SPECIFIED)) {
        return false;
    }
}
```

Please notice the following code:

```
static {
    Util.registerTransport();
}
```

This code needs to be added to the class implementing the query support or else you get an internal nested runtime SSL exception with no clue to where it is thrown, which is almost impossible to debug. But even with the above code the web service might fail with the information that it doesn't know the protocol "`https:`". This problem is solved by copying the `client-config.wsdd` file found in your Globus installation to the execution directory or in the root of the JAR file of your code.

5.4 Engineering Strategy

The following section is devoted to introduce a future maintainer to the engineering strategy used when designing and implementing the grid-aware SDK API discussed in this thesis. It is important to understand this strategy as a future maintainer is required to follow it.

The strategy is based on common practice based on my experience within the field of computer science plus additional practices applied here at the German Aerospace Center.

The specific development process used by RCE (and derived products) is described in the internal document "*The RCE Development Standard*" by Andreas Schreiber and Thijs Metsch [SESIS]. This above document lists the requirements briefly without much explanation to why these strategies are used. In this section I will try to compensate for that by explaining why I think these strategies make sense and by showing how they have been applied during the thesis period.

When building a long-living product within a team it is of outmost importance that a good engineering strategy is followed. A good strategy should help improving software quality, reduce costs and also facilitate easy maintenance of the software project. Some of the strategies we have chosen are listed below, and belong to good software engineering practices.

1. The design needs to be well thought out in order to limit design mistakes and to ease maintenance.

2. The design decisions needs to be documented so that other developers can participate or take over the project maintenance. This can also serve as requirement contract between development team and contractor.

3. The design needs to be modular and it should be possible to test as much of the code separately in order to limit and localize bugs and design mistakes. Here unit testing plays a role.

4. A control versioning system should be used to ease collaboration.

5. A bug-tracking system should be used in order to follow up on bugs, feature requests and milestones.

6. The code needs to be built regularly in order to catch when a change in one part of the system affects the system in another part.

7. The code and the API should be documented and documentation must be generated.

8. Coding standards must be used to enforce homogeneity, ensure clean API's and to ease code reading and understanding.

9. New code should be reviewed and tested.

10. Profiling should be used to find the performance bottlenecks.

In the following sections I will look further at the items listed above. I will describe why these points make sense and how they are used in the RCE/SESIS projects as well as for this thesis. Showing how these strategies have been used should help the future maintainer to better apply these during maintenance or further development.

5.4.1 Step 1: Design before Programming

When you are going to make a program, independently of project size, it is always good practice to think about the design of the project before actually starting the implementation.

Designing as you go along seldom leads to good design; even though you are a good designer you end up designing only parts of the programs at the same time, which often require changes to the rest of the design – changes that might be so large that redesign is needed. Humans generally don't have the ability to deal with big amounts of data at the same time, and keeping the overview of a program is basically impossible without having it well designed and having looked at the design abstractly.

When you plan your design well before coding, you catch many of your mistakes and as an added value, others can easily give comments on your design and when every thing is well documented it serves as good documentation that can help with maintenance.

Most often when designing you don't design every thing from scratch but you reuse code and libraries that others have designed. There are numerous reasons for doing this. First of all, you often save a lot of time since you don't have to code everything yourself, plus most of the time, the libraries you use are more maintained and well-tested that what you could produce yourself. The reason for this is that the people who have developed these libraries often have a lot of know-how about the technologies the libraries implement and what they have produced is most-often of a better quality that what you could produce with your knowledge. Basically, when using these libraries you get the following advantages:

- Your save time implementing this functionality
- Your save time fixing bugs and maintaining this code
- Your get higher quality code than what you normally could produce yourself
- You use well-tested code that is in use by more people that your code would be.

But, using code other people have produced can have problems as well. It is clear that the code could be badly maintained, badly written and full of bugs, not integrate into your project as well or simply have dependencies on other code that either conflict with your project or seem overkill for the task you are trying to solve.

Due to these facts a so-called Technology Study is often needed in the design phase. Depending on the wrong libraries and technologies can have catastrophic consequences for a project. Just imagine integrating the wrong technology in your project and realizing that the libraries buggy and not maintained. A case like the like this would often either make you're the maintainer of the buggy library (in case the source code is available) or will make you spend lots of time working around the bugs or simply redesigning the whole program to get rid of the dependency; something that can be very costly.

To avoid these problems with the Grid-integration and still not to design any grid middleware ourselves, a lot of attention during this project period has been given to learn about Grid Technologies so that we will make future-proof decisions so that we avoid the common pitfalls of depending on others code.

The choice of programming language, IDE, tools etc is also a part of the technology study, but these things have already been specified by the RCE/SESIS project, such that we use Java as programming language and everything is build on and with the Eclipse Platform.

5.4.2 Step 2: Document Design Decisions

As stated before 80% of the development cycle of a product is code maintenance, which definitely shows the importance of facilitating maintenance by providing good documentation.

Though, some will argue that source code serves as the best documentation, source code can be hard to understand as well, even by the author himself. This often has to do with the fact that the source code doesn't include information about the ideas behind the implementation but just shows the exact implementation. For this reason it is important to document source code to some extend and write down the consideration in a design document. This way a future maintainer can understand the implementation and pick up the work where it is left, fix bugs etc. When you don't understand the code it is easy to get tempted to rewrite it or throw away code paths that you don't understand to later find out the code was there for a reason.

The general object oriented design should be documented by using UML diagrams, as that is the standard way of doing this today and because they provide a good overview over the implementation. Algorithms, design decisions and the like should, depending on their size and relevance, be either documented in the source code or in the design document.

In this project, this thesis serves as documentation for the implementation. It explains the technologies used and shows UML use-case and class diagrams for the implementation. No particular hard-to-understand algorithms have been used, and thus most documentation in this area is found in the source code. On the other hand, the grid services used to implement the functionality needed has been documented in this paper, in section 5.3.

5.4.3 Step 3: Design modularly

The idea behind designing modularly is to organize a (complex) system into a set of components that in fact can be developed independently and later be put together to form a solution to a problem.

The basic idea is to only put related code into the same class and define simple interfaces for all interactions. These interfaces can make it easier to reuse components and reduce the number of interactions that needs to be checked when verifying that a component works as intended.

In our design all parts that doesn't directly has something to do with each other has been separated into different classes implementing different interfaces. This makes it easier to understand the code and it also means that if we for instance need to change the data management features to use another grid service, we would only need to make these changes to one class, or we could even just implement another implementation of the interface and let it up to the use to decide which one to use.

The use of interfaces is of great importance to the grid-aware API as we need to be able to support more than one grid middleware solution. The benefit from using modularity doesn't just come from subdividing the program but also from information hiding. It should be clear that the way a program is subdivided has an impact on the ability to implement and modify the program. By using information hiding it is possible to hide information in modules that should not be available to the rest of the program. With regard to the grid-aware API, it is clear that the Globus implementation specifics should be encapsulated and not be available to the rest of the program, nor the UNICORE implementation. Using interfaces, it is possible to use one API and have the backend (Globus or UNICORE) change without having to change the program using the API. Effort has been put into make the design of the grid-aware API so that no implementation should require changes to the current interfaces.

The question is then what to encapsulate. Generally it is important to encapsulate functionality that is likely to change later or that needs different implementations such as in our case. Of course, encapsulation can also be used to reuse code that occurs in various places and in that way form a common implementation. Generally, it can be said that code that can be reused should be encapsulated.

We can ask a few questions to check if we have modulated our design enough. The design presented in this thesis passes these questions.

- The design consists of clearly defined modules
- The purpose of the modules are clearly defined
- The different modules do not duplicate functionality
- All aspects that are likely to change have been isolated
- Is it not useful to subdivide the modules further
- The interfaces do not expose implementation details

5.4.4 Step 3: Make use of Unit Testing

Testing code is important and the more that can be done automatically, the better. For this particular reason we have decided on using unit tests when implementing RCE and the grid support.

A unit test is a methods used to test if a particular function is working as intended. The idea is to write test cases (different input, etc) for all functions so that all regressions can be caught automatically and can be identified and fixed before they cause problems elsewhere in the software project. This way time can be saved.

It also welcomes change as it allows refactoring code while still being sure that the module works as intended due to the automatically regression testing. Additionally, the tests serve as some kind of API documentation, as other developers can look at the unit tests to gain basic understanding of the API. Though, ordinary documentation is more often better written and more understandable than reading source code, documentation tend to become outdated, where as the unit tests will always follow the implementation.

For the project we have used JUnit [JUnit] for implementing unit tests, as it works with Java source code and integrates well with Eclipse.

There have been added unit tests for all classes implemented. Since most unit tests are similar, most unit tests can be based on the following example:

```
/*
 * Standard header.
 */

package de.rcenvironment.rce.packagename;

import junit.framework.TestCase;

/**
 * Test cases for the class <code>ClassName</code>.
 *
 * @version $LastChangedRevision$
 * @author Name of Author
 */
public class ClassNameTest extends TestCase {

    /**
     * Declaration of fields and mock objects.
     */
    private static final long SOME_FIELD = 86400000L;

    /**
     * The class under test.
     */
    private ClassName myClassName = null;

    protected void setUp() throws Exception {
        super.setUp();
        // Setting of fields, needed objects and mock objects.
        myClassName = new ClassName();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
        // Cleaning up.
        myClassName = null;
    }

    /*
     * #################### Test for success ####################
     */

    /**
     * Test <code>.setValueMethod()</code> for success.
     *
     * @throws Exception if exception occurs.
     */
    public void testSetValueMethodForSuccess() throws Exception {
        myClassName.setValueMethod(var1);
    }

    /*
     * #################### Test for sanity ####################
     */

    /**
     * Test <code>.setValueMethod()</code> for sanity.
    public void testSetValueMethodForSanity() {
        final int amount = 23;

        myClassName.setValueMethod(amount);

        int amountStored = Integer.parseInt(myClassName.getValueMethod());

        assertEquals(amount, amountStored);
    }

    /*
     * #################### Test for failure ####################
     */

    /**
     * Test <code>.setValueMethod()</code> for failure.
     */
    public void testSetValueMethodForFailure() {
        try {
            myClassName.setValueMethod(12312432);
            fail("Should throw a OutOfRangeException.");
        } catch (OutOfRangeException e) {
            assertTrue(true);
        }
    }
}
```

As it can be seen from the example code, the idea is to test all methods three times. First for success, then for sanity, and last but not least, for failure.

When you test for success you call the function and if something doesn't work internally, as for instance if it throws a `NullPointerException` it will fail and caught by the unit tests. When you test for sanity, you try testing if the function works the way it is supposed to. In the example I set a value and I check if the right value is set. When I test for failure, I give values that should result in failure, as for instance result in the fact that an exception is thrown.

It should be noted that 100% code coverage (test functions for each method) is not possible, as it is impossible to test some methods as it either doesn't make sense or because they need a running system. Instead these should later be tested by integration tests, running real code.

It is good practice to start by implementing the unit tests before actually implementing the design, as it forces the developer to think about the design and how the methods really should work. Since many classes will not be available at this time, it is often possible to use so-called mockup objects, which are fake objects acting like the real one. Since there are good resources on mockup objects online, I won't go more into detail here.

To round of this section I will look at one of the unit tests implemented during the project, as this will give a real life example. The test in question is the `GlobusConnectorTest`. All comments have been removed to keep it concise.

```
package de.rcenvironment.rce.grid.globus;

import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;
import junit.framework.TestCase;

import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;

public class GlobusConnectorTest extends TestCase {

    private static final long PERIOD_OF_VALIDITY = 86400000L;
    private final String myHost = "127.0.0.1";
    private final int myPort = 80;
    private Date myTimestamp = null;
    private X509Certificate myX509Certificate = null;
    private ProxyCertificate myProxyCertificate = null;

    private GlobusConnector myGlobusConnector = null;

    protected void setUp() throws Exception {
        super.setUp();
        InputStream inputStream =
            getClass().getResourceAsStream("/usercert.pem");
        myX509Certificate = CertUtil.loadCertificate(inputStream);

        myTimestamp = new Date();
        myTimestamp.setTime(myTimestamp.getTime() +
            PERIOD_OF_VALIDITY);

        myInvalidTimestamp = new Date();
        myInvalidTimestamp.setTime(myTimestamp.getTime() -
            PERIOD_OF_VALIDITY);

        myProxyCertificate = new ProxyCertificate(myX509Certificate,
            myTimestamp);
        myInvalidProxyCertificate = new ProxyCertificate(myX509Certificate,
            myInvalidTimestamp);

        myGlobusConnector = new GlobusConnector();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
        myProxyCertificate = null;
        myGlobusConnector = null;
    }

    public void testAuthenticateForSuccess() throws Exception {
        myGlobusConnector.authenticate(myHost, myPort, myProxyCertificate);
    }

    public void testAuthenticateForSanity() throws Exception {
        try {
            myGlobusConnector.authenticate(myHost, myPort,
                myInvalidProxyCertificate);
        } catch (UserException e) {
            assertTrue(true);
        }
    }

    public void testAuthenticateForFailure() throws Exception {
        try {
            myGlobusConnector.authenticate(myHost, myPort, null);
            } catch (SystemException e) {
                assertTrue(true);
            }
        }
}
```

To test the unit tests by using Eclipse, you right click on the package name containing the tests and choose Run As/JUnit Test, as illustrated in the following figure.

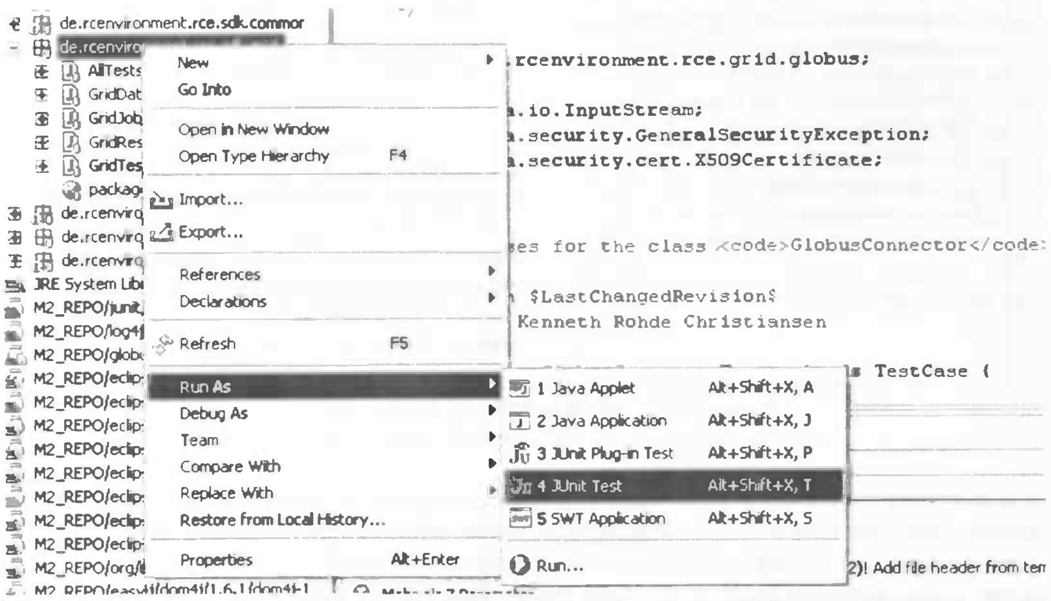
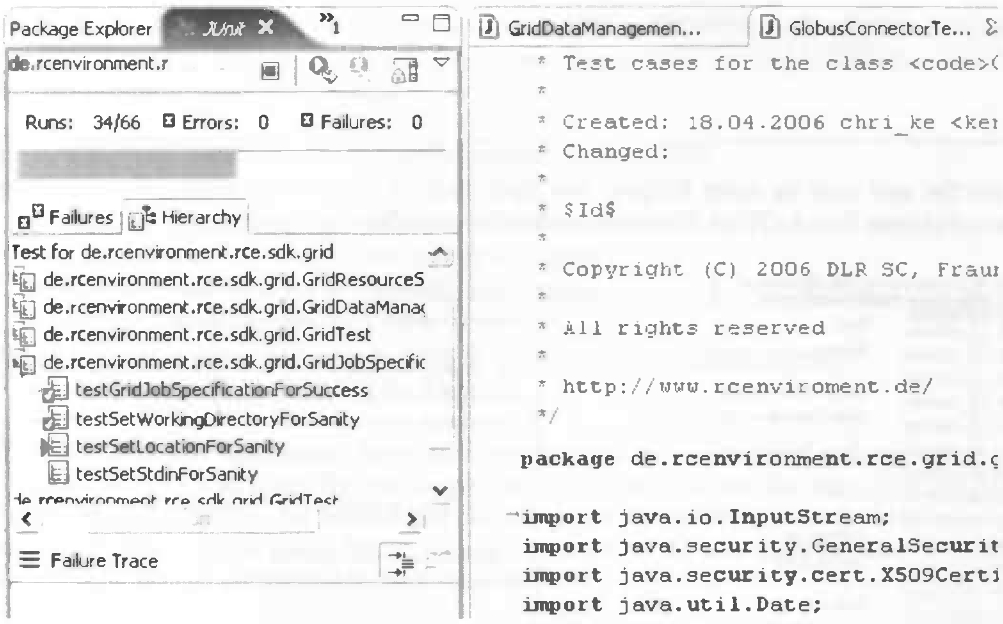

```
.rcenvironment.rce.grid.globus;

a.io.InputStream;
a.security.GeneralSecurityException;
a.security.cert.X509Certificate;

es for the class <code>GlobusConnector</code>

 $LastChangedRevision$
 Kenneth Rohde Christiansen

s TestCase {
```

Figure: Running a JUnit unit test suite from within Eclipse.

The following illustration shows a test suite in the running. When a unit test fails, the JUnit Eclipse plugin will show you the failure trace that makes it easy to pinpoint and fix the mistake in the code.



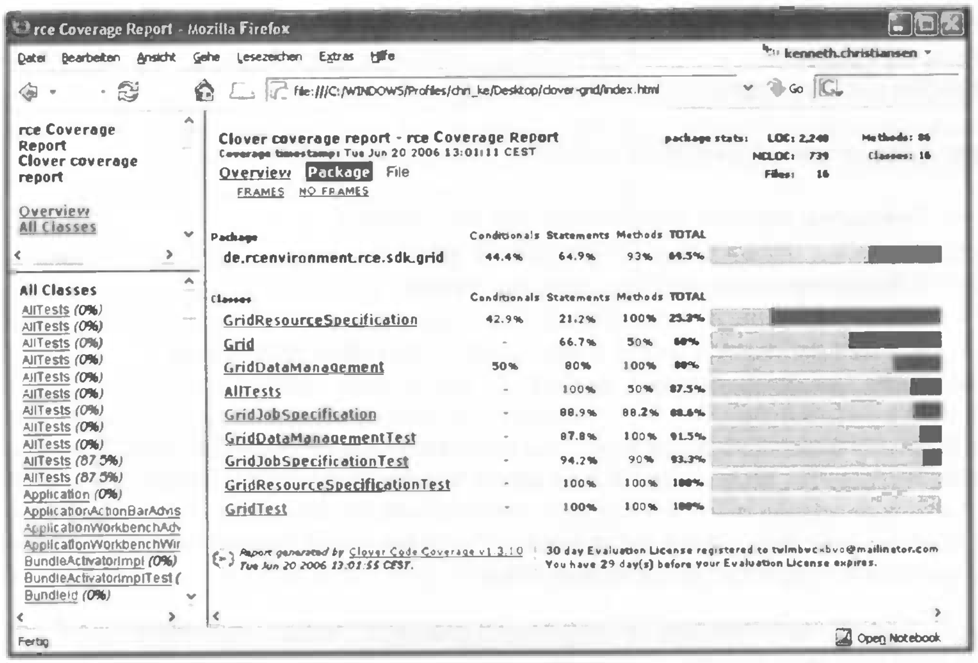
Runs: 34/66 Errors: 0 Failures: 0

```
Test for de.rcenvironment.rce.sdk.grid
de.rcenvironment.rce.sdk.grid.GridResourceS
de.rcenvironment.rce.sdk.grid.GridDataManag
de.rcenvironment.rce.sdk.grid.GridTest
de.rcenvironment.rce.sdk.grid.GridJobSpecific
    testGridJobSpecificationForSuccess
    testSetWorkingDirectoryForSanity
    testSetLocationForSanity
    testSetStdinForSanity
de.rcenvironment.rce.sdk.grid.GridTest
```

```
 * Test cases for the class <code>(
 *
 * Created: 18.04.2006 chri_ke <kei
 * Changed:
 *
 * $Id$
 *
 * Copyright (C) 2006 DLR SC, Fraui
 *
 * All rights reserved
 *
 * http://www.rcenviroment.de/
 */
package de.rcenvironment.rce.grid.g

import java.io.InputStream;
import java.security.GeneralSecurit
import java.security.cert.X509Certi
import java.util.Date;
```

Figure: Illustrating a unit test suite being run.

It is one thing having the goal of using unit tests and having all unit tests showing green, but unfortunately this only tells little whether the test actually tests the code base and if not, what is not being tested. In order to make sure that a test suite doesn't get out of date, it is possible to use a code coverage analysis tool, and for the RCE/SESIS projects we deploy the Cenqua Clover tool that analyses the source code and generate coverage reports either as HTML or PDF. An example of the HTML output is shown below.



Clover coverage report - rce Coverage Report
Coverage timestamp: Tue Jun 20 2006 13:01:11 CEST

package stats: LOC: 2,240 Methods: 86
NCLOC: 739 Classes: 16
Files: 16

Package	Conditionals	Statements	Methods	TOTAL
de.rcenvironment.rce.sdk.grid	44.4%	64.9%	93%	69.5%

Classes	Conditionals	Statements	Methods	TOTAL
GridResourceSpecification	42.9%	21.2%	100%	25.3%
Grid	-	66.7%	50%	60%
GridDataManagement	50%	80%	100%	80%
AllTests	-	100%	50%	87.5%
GridJobSpecification	-	88.9%	88.2%	88.6%
GridDataManagementTest	-	87.8%	100%	91.5%
GridJobSpecificationTest	-	94.2%	90%	93.3%
GridResourceSpecificationTest	-	100%	100%	100%
GridTest	-	100%	100%	100%

Report generated by Clover Code Coverage v1.3.10 Tue Jun 20 2006 13:01:55 CEST.
30 day Evaluation License registered to twlmbwcxbvo@mailinator.com
You have 29 day(s) before your Evaluation License expires.

Figure: Illustrating Clover code coverage analysis tool

5.4.5 Step 4: Use of a Control Versioning System

Version control systems are indispensable when a team of developers are involved. When used in collaboration the developer always work on a local copy of the project, and is not bothered by the changes made by co-workers.

Once in a while the developer can check out the changes made by the co-workers from a so-called repository and in that way get their latest features, bug fixes etc, but is able to do so when he want to and has the time to check if the changes interfere with his code. When satisfied with his code, he can check it in into the main repository and others will get his changes next time they do an update.

When making large architectural changes it is possible to make a branch of the code tree, so that all the check-ins and check-outs will be from this "snapshot" of the project and thus not interfere with the main development branch.

It is also possible to tag special states of the code, for instance to mark that the state of the code corresponds to a release or milestone.

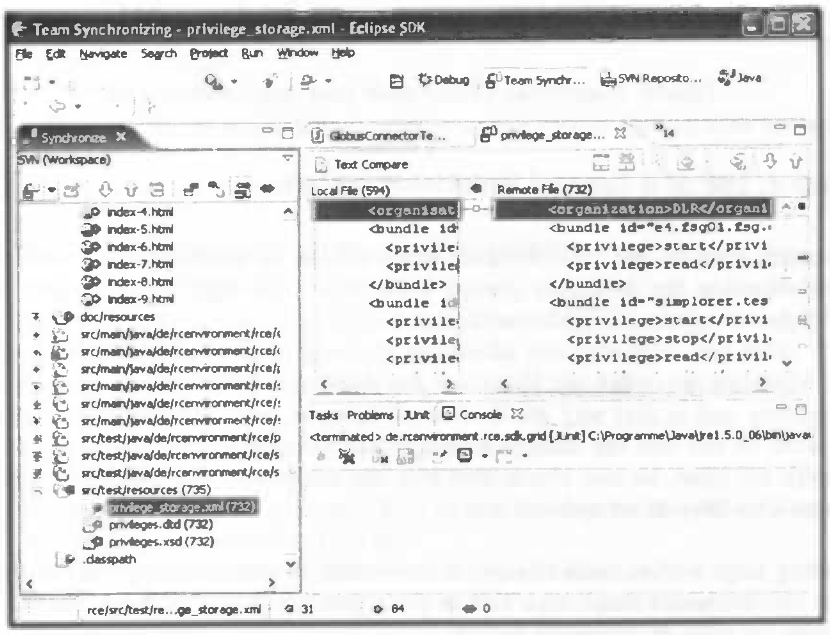
There are various commercial and non-commercial version control systems, and the most recognized one is the, now very old, CVS [CVS], which has native integration in our development platform Eclipse. During the last couple of years the open-source version control system Subversion [Subversion] is gaining popularity due to its features and that it is maintained and further developed.

For the RCE/SESIS project we will be using subversion in combination with the Subclipse [Subclipse] Eclipse plug-in, which provides subversion integration very much alike the CVS integration already presents in Eclipse.

Some of the features of subversion that differentiate it from CVS:

- Directories, renames, and file meta-data are versioned.
- Commits are truly atomic and not per file based.
- Efficient management of branching and tagging
- Efficient handling of binary files
- Has well defined API so that it can be called from other applications.
- Keeps metadata attached to the files.

The following illustration shows how team synchronization is performed with the use of the Subclipse plugin for Eclipse. By clicking on the Synchronize button, the latest changed from the Subversion repository are fetched and compared with the local copy. At the screenshot we see three numbers in the status bar indicating that there are 31 new updated files, 84 locally changed files not yet committed and no conflicts.


Team Synchronizing - privilege_storage.xml - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Debug
Team Synchr...
SVN Reposito...
Java
Synchronize
GlobusConnectorTe...
privilege_storage...
Text Compare
Tasks Problems JUnit Console

SVN (Workspace)

```
index-4.html
index-5.html
index-6.html
index-7.html
index-8.html
index-9.html
doc/resources
src/main/java/de/rcenvironment/rce/
src/main/java/de/rcenvironment/rce/
src/main/java/de/rcenvironment/rce/
src/main/java/de/rcenvironment/rce/
src/main/java/de/rcenvironment/rce/
src/main/java/de/rcenvironment/rce/
src/test/java/de/rcenvironment/rce/p
src/test/java/de/rcenvironment/rce/s
src/test/java/de/rcenvironment/rce/s
src/test/resources (735)
privilege_storage.xml (732)
privileges.dtd (732)
privileges.xsd (732)
.classpath
```

Local File (594)

```
    <organisat
    <bundle id
      <privile
      <privile
    </bundle>
    <bundle id
      <privile
      <privile
      <privile
```

Remote File (732)

```
    <organization>DLR</organi
    <bundle id="e4.fsg01.fsg.
      <privilege>start</privi
      <privilege>read</privil
    </bundle>
    <bundle id="simplorer.tes
      <privilege>start</privi
      <privilege>stop</privil
      <privilege>read</privil
```

```
<terminated> de.rcenvironment.rce.sdk.grid [JUnit] C:\Programme\Java\jre1.5.0_06\bin\java
```

```
rce/src/test/re...ge_storage.xml   31   84   0
```

Figure: Team Synchronization with Subclipse and Subversion

In the case of conflict it is possible to click on the conflicted file and compare the local file with the one in the repository and then merge in the changed manually.

For implementing the grid support to RCE no branch was created as work could be done locally and because the implementation was separate from most other functionality in RCE. This means that the only was the grid support could disturb the other team developers were if the code didn't compile, which theoretically cannot happen as the code is compiled and tested on check-in.

5.4.6 Step 5: Use of an Issue Tracking System

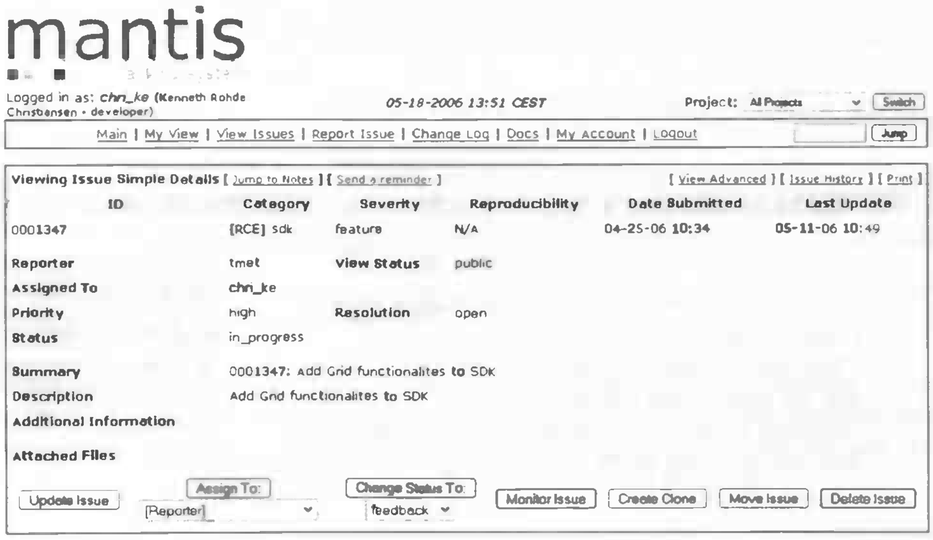
Every software project contains bugs. It is simply practically impossible to program without making mistakes and therefore there is a need to track these bugs. Tracking bugs can be difficult; most projects and bugs are long living and most of the time the bugs are quite complex and require extra information in order to solve them.

Issue tracking includes recording the bug, confirming it, getting additional information, discussing how to fix it, and then deciding whether or when the bug should be fixed considering importance, plus time and budget to fix the bug.

It is quite obvious that keeping good record of the bugs, the discussions and the decisions made will help a lot in a software project. Issue tracking systems exists for among other this purpose and they enable you to attach info to each bug issue such as assigned developer, discussion details, etc, as well as support finding bugs by different means such as searching.

Issue tracking systems can also be used for other things than tracking the status of bugs; for instance it can be used for feature tracking, hence the name Issue Tracking System.

For the SESIS/RCE project the open-source issue tracking system Mantis [Mantis] is used. The system is also used for tracking enhancement requests, and bugs can also be marked as such.


mantis
Project: All Projects
Switch
Main | My View | View Issues | Report Issue | Change Log | Docs | My Account | Logout
Jump
[Jump to Notes] [Send a reminder]
[View Advanced] [Issue History] [Print]
Update Issue
Assign To:
Change Status To:
Monitor Issue
Create Clone
Move Issue
Delete Issue

Logged in as: chn_ke (Kenneth Rohde Christensen - developer) 05-18-2006 13:51 CEST

Viewing Issue Simple Details

ID	Category	Severity	Reproducibility	Date Submitted	Last Update
0001347	[RCE] sdk	feature	N/A	04-25-06 10:34	05-11-06 10:49

Reporter	tmet	View Status	public
Assigned To	chn_ke		
Priority	high	Resolution	open
Status	in_progress		
Summary	0001347: Add Grid functionalites to SDK		
Description	Add Grid functionalites to SDK		
Additional Information			
Attached Files			

[Reporter] feedback

Figure: The mantis report used for tracking the implementation of the Grid SDK functionality.

The above illustration shows the report used for tracking the implementation of the grid SDK support. It can be seen who the issue/report is assigned to and what the status are.

The issue tracking system has been used during the implementation and each time something has been committed to the subversion repository a message entry was added to a report concerning the implementation. This is the policy for the RCE project. Additionally, new reports are opened to triage bugs and to fix the issues found during code review.

Due the lifecycle of an issue, the issue is represented by a certain status. When a new issue is added it has the status "new" that can be "confirmed" by or "assigned" to a developer. If the issue is not obvious the status can be set to "feedback". When the "assigned" developer starts working on an issue the status is changed to "in_progress". When an issue is resolved the status can be changed to "resolved" and when it has been reviewed by a tester or technical project leader it can finally be "closed". The whole process looks like the following:

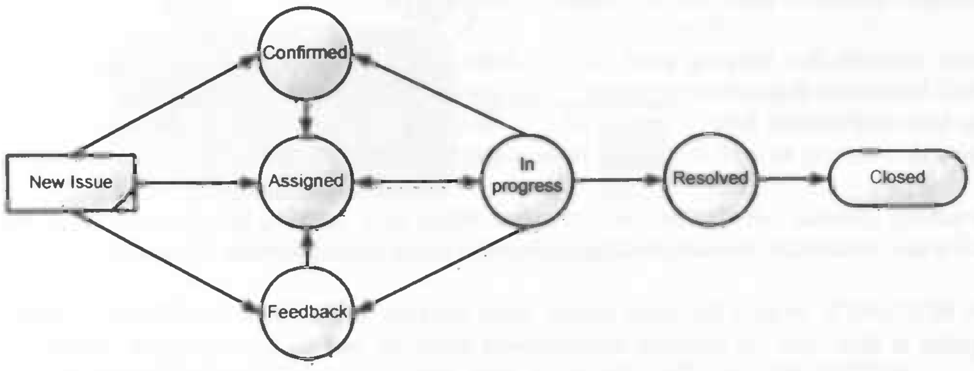


Figure: The issue states in the Mantis issue tracking system deployed for the project.

5.4.7 Step 6: Build regularly using a Build System

Building a product can be a very complicated task. Most often it is not just enough to compile a few files, but often you need to generate additional resource files and make it all easily distributable. A typical Java based application goes though the following steps:

- Compile the sources with Java compiler of choice.
- Copy resource files into the right directories.
- Clean up and remove generated files so that the next build will be a clean build.
- Zip the project into a Java JAR file.
- Generate JavaDoc information from the source code.
- Run the unit tests to see if everything is still working as intended.
- Deploy the product.

In order to do the above, most Java projects today deploy the Apache Ant XML-based build scripting system that allows you to automate the above-mentioned steps. Unfortunately, Ant is not perfect as it contains no convenient ways of reusing the scripts which means that developers often tend to copy and paste parts of the scripts. Another problem is that Ant doesn't handle dependencies, which means that the developers often check the required JAR files into their source code repository, something that the source code repository is not really meant for.

For the RCE/SESIS project we deploy Maven [Maven] as a build system. Maven is developed by the Apache Foundation, which is also the case for Ant. In comparison to the features of

Ant, Maven additionally handles dependencies. When introducing such new dependencies as the CoG Toolkit, this dependency is added to the Maven project configuration and the dependency will automatically be downloaded to the developers when Maven is run locally.

Maven works with Project Object Model (POM) files for describing a project. You simply describe your project by supplying information about directory structure, dependencies, programmers etc, and then Maven takes care of the rest.

Adding a dependency, like for instance the `cog-jglobus.jar` distributed with the CoG Toolkit, is done by adding the following code to the POM file, describing the whole RCE/SESIS project.

```
    <dependency>
      <groupId>globus</groupId>
      <artifactId>cog-jglobus</artifactId>
      <version>1.2</version>
      <scope>compile</scope>
    </dependency>
```

This code will first look in the SESIS repository for a file called `cog-jglobus-1.2.jar` located in the `globus` directory. If not found, it will look in the repository http://www.ibiblio.org/maven2 . This repository contains the most commonly used jar files, so it is always a good idea to look there before adding new JAR files to the SESIS repository.

When developers are working together they tend to often break each others code, by introducing changes that affects other parts of the system. In order to keep this under control automatically builds can be used.

An automatic build systems simply checks out the code from a repository and tries building it (for instance on several different systems to check portability) and when something goes wrong it will identify where (build logs) and the state of the files that made up the build so that it can find out who broke the build and then inform this developer.

In this way you hold the developers accountable for their actions and force them to either fix their changes quickly or take them out.

For doing the above we are deploying the open source CruiseControl product, which is a framework for a so-called continuous build process, which is what we can automatic builds. It provides a web interface for looking at the success or failure for the current and previous build, and includes plugins for notifying the developers of build breakage.

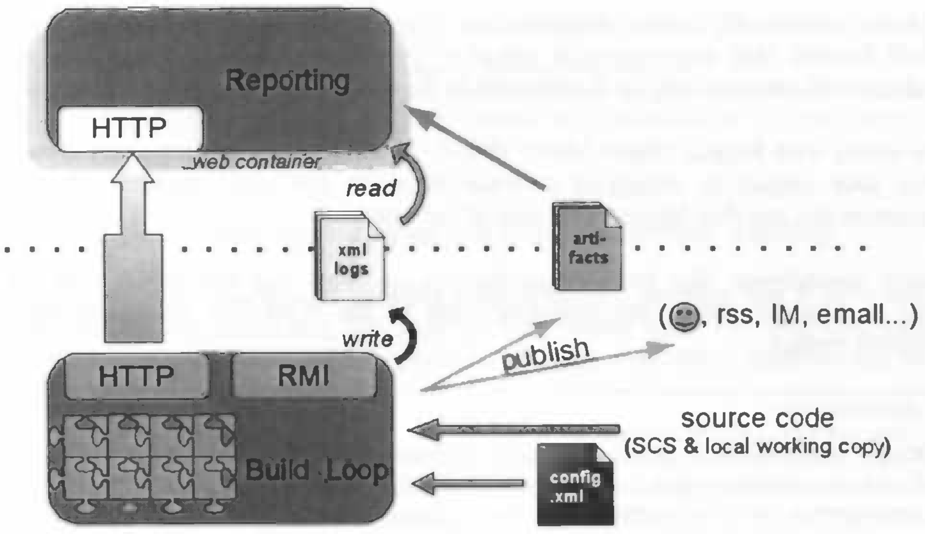


Figure: The architectural design of the CruiseControl automatic build system.[37]

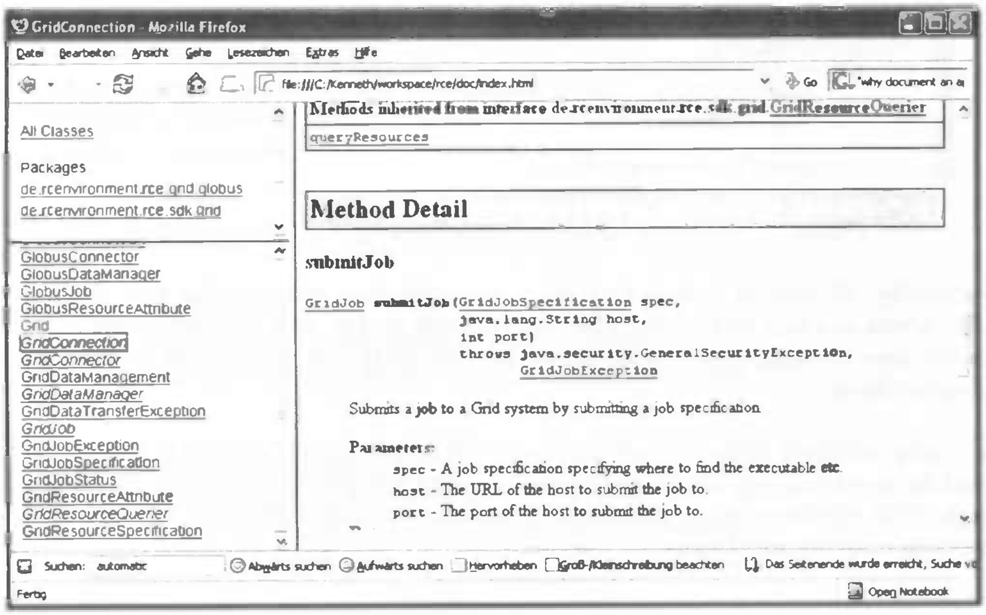
5.4.8 Step 7: Document the Code and generate API Documentation

As mentioned before design decisions and algorithms and the like should be documented in design documents. On the other hand, this doesn't mean that the source code never should be documented.

The source code is the exact implementation and often implementation decisions are being taken which as such are so specific that they don't belong in the design documentation. Most of the time source code written with a good coding style is easy readable, but when things might be unclear to future maintainers or even to the developer himself when he has to continue working on the code the day after, it is definitely a good idea to add a comment. But adding comments can also make the code less readable as too many comments distract the reader from the real code.

It is good practice to document all functions and class fields with Javadoc and it is even a requirement for the RCE project. This way the API will be documented and the comments will mostly be limited to particular places in the code. Other comments should be used when they make sense but not be overused. This is the strategy we have applied for this project.

[37] From: http://cruisecontrol.sourceforge.net/overview.html



Methods inherited from interface de.rcenvironment.rce.sdk.grid.GridResourceQuerier

queryResources

Method Detail

submitJob

```
GridJob submitJob(GridJobSpecification spec,
                  java.lang.String host,
                  int port)
           throws java.security.GeneralSecurityException,
                  GridJobException
```

Submits a job to a Grid system by submitting a job specification

Parameters:

spec - A job specification specifying where to find the executable etc.
host - The URL of the host to submit the job to.
port - The port of the host to submit the job to.

Figure: Browsing the Javadoc documentation generated from the Grid source code.

Generally it is a good idea to document an API, as Javadoc can be generated automatically and later serve as API documentation. The Javadoc comments can also be used by Eclipse directly and show you the comments in tooltips when using the methods in question.

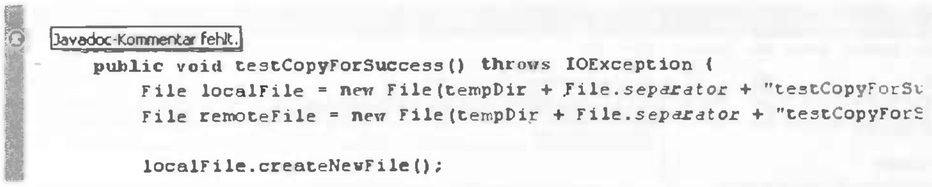
5.4.9 Step 8: Enforce Coding Standards

Even though you might know almost everything about a particular programming language, it is still quite possible to write programs that are unreadable, difficult to debug and hard to maintain; and not even just by others but by your self as well. The impact of coding style goes beyond the individual as programming is often team work. If every team member has his/her own approach to documenting, structuring, naming; every piece of the program quickly becomes it own little isle, and it becomes harder for others to step in and perform work in this area without hitting the wall.

When a consistent approach is use when developing programs, the code becomes a lot easier to read and thus to maintain. Experience here at DLR (German Aerospace Center) confirms that 80% of the lifetime of a software product is maintenance; a maintenance done by different people over time.

For the RCE platform coding standards have been specified at German Aerospace Center which standardizes among others, naming conventions and code layout. All new code produced should then try to conform to these standards.

In order to secure that the coding standards are followed, some standard settings are imported in Eclipse and a plug-in called Checklipse [Checklipse] is used that automatically checks the coding style following some given rules.



```
Javadoc-Kommentar fehlt.
    public void testCopyForSuccess() throws IOException {
        File localFile = new File(tempDir + File.separator + "testCopyForSu
        File remoteFile = new File(tempDir + File.separator + "testCopyForS

        localFile.createNewFile();
```

Figure: Checkclipse indicating that a Javadoc comment is missing.

Additionally, all code is reviewed by other team members ensuring that it lives up to the quality levels and as a part of this, also uses the right coding style. The reviewing process is also less time consuming due to the improved readability that comes from the use of these coding standards.

The coding standards defined by my co-worker Thijs Metsch can be found in the appendix. It should be noted that only source code files following these coding standards can be checked into the RCE repository as the files are automatically checked on check-in, and rejected if not complying with this set of rules.

5.4.10 Step 9: Review and test new Code

It is generally known in most software companies that code reviews reduces development time and ensures quality. Some of the benefits of code reviews are listed below:

- Catches bugs earlier in the development cycle. The earlier bugs are found in a development cycles the cheaper and easier they are to fix.
- Developers have the opportunity to learn from the more experienced peers. You can learn a lot from reading other people's code.
- When you know someone else is going to look over your code you are more likely to tidy it up, document it and make sure that you are not making embarrassing mistakes.
- It helps to avoid common mistakes, as others are likely to find mistakes that you missed.
- Creates communication between the development team and gives you a better overview over the whole project.
- The process of explaining your code to others helps you actually reviewing the code yourself, as you for once is not just looking and seeing what you expect to see.
- It is an insurance against people leaving the company, leaving code behind that only they can understand and maintain.
- In general, it helps saving time and producing higher quality code that is easier to maintain.

Unfortunately, most fellow developers are often buried in work and under deadline pressure, which results in the fact that reviews are often skipped or only done superficies as people just "bet" that there aren't really any problems. A quick review is though; always better than no review at all!

At German Aerospace Center code reviewing is a mandatory part of the development process and thus, get all the advantages. In order to help the reviews a list of possible questions has been made:

The big four
Is it possible to understand the source code?
Has there been made unit tests for the source code?
Have the changes been documented?
Does the change implement a feature or does it fix a bug?

Coding Standards
Does the code follow the given coding standards?

Is everything documented?
Does there exist so-called dead-code (classes, methods, variables)
Does the documentation of a method show which parameters it works with?

Design
Is it possible to understand the design?
Does the implementation follow the design specification?
Is all the functionality of the design specification actually implemented?

Maintainability
Are all the comments necessary or are they just reducing the readability?
Do the comments fit?
Are all variables, classes and methods written correctly?
Are all variables documented (with domain, size and limits?).

Documentation
Are command line arguments and environment variables documented?
Are (all / public) methods documented?
Does the implementation follow the design specification?
Have all changes been documented for instance by the use of a ChangeLog?

5.4.11 Step 10: Use Profiling to find Performance Bottlenecks

It is generally known that you shouldn't optimize while coding as the code often becomes more unreadable and most of the time you don't gain anything from the optimization. The problem is that the optimization might be useless if there is another performance bottleneck in the code path. It is practically impossible for a programmer to know where the bottlenecks are and because of this reason code profiling is often used.

Code profiling means determining how often certain pieces of code are executed. By knowing the frequency with which a piece of code is in use, you can more accurately determine the importance of optimizing that particular piece of code.

Code profiling can also be used the other way around, meaning that when you are cleaning up code and doing refactoring, it is possible to check whether is has a positive or negative impact on the performance.

Code profiling has not been done on the grid implementation. The reason is that it doesn't make much sense at this point as we still don't have any real life extension methods taking advantage of the code.

Chapter 6

Examples and Demo of the Grid Support

The thesis wouldn't be complete without at least showing how the added grid support can be put to use. A lot of attention has been put to make the API so easy to use as possible as the examples in this section will show.

I will start out with a very simple example of executing the standard UNIX command "yes" on a grid system. The program "yes" is simply a program that continues printing the argument that you give to it, which has some practical use when working with the UNIX command line. Though, this has no practical use on a grid system, it shows very well how to get around with the API. Furthermore, the integration tests are good examples of how to use the grid-aware SDK, and have all been put in an appendix.

Before we can use the grid system we need to get a RCE proxy certificate. When incorporating grid resources into an extension method, you should already have such one available, but if not, the following code will read an X509 certificate from the hard drive and generate one for us.

```
/**
 * Read the user certificate.
 *
 * @return a proxy certificate for the user.
 * @throws UserException if the certificate is invalid.
 */
public static ProxyCertificate getUserCertificate()
    throws UserException
{
    InputStream inputStream = new
        Object().getClass().getResourceAsStream("/usercert.pem");
    X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

    Date timestamp = new Date();
    long periodOfValidity = 86400000L;
    timestamp.setTime(timestamp.getTime() + periodOfValidity);

     return new ProxyCertificate(x509Cert, timestamp);
}
```

The code should be pretty self explanatory as it simply reads in a permission (X509 certificate) file as a stream, generates an X509Certificate from it and converts it to a RCE proxy certificate after having specified a time-to-life value.

Though, not really necessary for our example, it is possible setting grid resource requirement, such as operation system version, amount of memory needed etc. Let's try setting a few requirements.

```
/**
 * Set the Grid resource requirements that we need for the
 * "yes" command.
 *
 * @return a resource specification or
```

```
public static GridResourceSpecification getResourceSpecification()
{
    GridResourceSpecification resspec = new GridResourceSpecification();

    resspec.setNoOfProcessors(1); // We just need one CPU
    resspec.setMinimumMemory(16); // We require 16 MB

    // Lets set some more specific attributes.
    resspec.addOptionalAttribute(GridResourceAttribute.MACHINE_TYPE,
        "i386");
    resspec.addOptionalAttribute(GridResourceAttribute.OS_NAME, "linux");

    return resspec;
}
```

The API features methods for specifying the most used requirements, but it is still possible adding more specific requirements, using the `addOptionalAttribute()` method. In the above code sample I set the number of processors to one, the minimum memory requirement to 16 MB and we specify that I need a Linux system running of an i386 architecture.

How we have almost all the basic things needed to submit a job. All we are missing is specifying the job specifics such as where the executable is located, output direction etc. As a minimum you always need to specify the location of the executable. The other specifications are optional.

```
GridJobSpecification jobspec = new GridJobSpecification();

jobspec.setResourceSpecification(getResourceSpecification());

jobspec.setWorkingDirectory("/usr/bin/");
jobspec.setLocation("yes");
jobspec.addArgument("hi");
jobspec.setStdout(System.out); // Use standard out
```

In the above we set the location to where the "`yes`" command can be found on our Linux system, and we add the argument "`hi`". Additionally we set the standard output to `System.out`, so that we get the output of the yes command written on the console.

The output should be:

```
$ hi
$ hi
$ hi
 ...
```

In order to submit the job, we need to authenticate with the Grid middleware running on our client. We do this by the following piece of code:

```
GridConnection conn = Grid.authenticate(cert);
```

The above code can throw two exceptions. A `SystemException` if the user certificate couldn't be found, and a `UserException` if no authentication could be made with the underlying grid middleware.

Now that we have a `GridConnection` object we can submit our job and play a bit around with it.

At home I have a Globus 4 installation running on the machine "`dheghnom.dnsalias.com`", so in the example I will submit the job to this host. The following code submits the job – which is in fact an infinite loop writing "hi". We then suspend the job one time and later resume it.

```
// Run job on the same machine.
GridJob job = conn.submitJob(jobspec, "dheghnom.dnsalias.com", 8443);

System.out.println("Going to suspend the job.");
job.suspend();
System.out.println("Status is: " + job.getStatus().toString());

System.out.println("Let's start the job again.");
job.resume();
System.out.println("Status is: " + job.getStatus().toString());
```

Running the example program yields the following output. "…" specify output removed.

```
 …
$ hi
$ hi
$ hi
 …
$ hi
 …
$ Going to suspend the job.
$ Status is: SUSPENDED
$ Let's start the job again.
$ hi
 …
$ hi
$ Status is: RUNNING
$ hi
$ hi
 …
```

As it can be seem from the example, RCE now offers a simple-to-use API for dealing with grid resources. Apart from the features shown in this simple example, the API also offers functionality for querying resources (ensuring that resources are there) and staging data files.

Since not all hardware resources are tested with job submission we can use the querying to insure this. So in order to insure the requirements we sat we add the following test

```
GridResourceSpecification resspec = getResourceSpecification();

if (!conn.queryResources(resspec, "dheghnom.dnsalias.com", 8443)) {
    System.out.println("Grid node is not fulfilling resource " +
       "the requirements");
    return;
}
```

The `main` function from the example, together with error handing is shown here:

```
/**
 * Play around with the Linux "yes" command by running it
 * on a Grid node.
 */
public static void main(String[] args)
{
    ProxyCertificate cert;

    try {
        cert = getUserCertificate();
    }
    catch (UserException e)
    {
        System.out.println("Failed getting user certificate");
        return;
    }

    GridConnection conn = null;
    try {
        conn = Grid.authenticate(cert);
    } catch (SystemException e) {
        System.out.println("Couldn't not find the user certificate");
        System.out.println(e.getMessage());
        return;
    } catch (UserException e) {
        System.out.println("Couldn't not authenticate with underlying "
            + "Grid middleware");
        System.out.println(e.getMessage());
        return;
    }

    GridJobSpecification jobspec = new GridJobSpecification();
    GridResourceSpecification resspec = getResourceSpecification();

    jobspec.setResourceSpecification(resspec);

    jobspec.setWorkingDirectory("/usr/bin/");
    jobspec.setLocation("yes");
    jobspec.addArgument("hi");
    jobspec.setStdout(System.out);    // Use standard out

    if (!conn.queryResources(resspec, "dheghnom.dnsalias.com", 8443)) {
        System.out.println("Grid node is not fulfilling resource " +
            "the requirements");
        return;
    }

    try {
        GridJob job = conn.submitJob(jobspec, "dheghnom.dnsalias.com",
            8443);

        System.out.println("Going to suspend the job.");
        job.suspend();
        System.out.println("Status is: " + job.getStatus().toString());

        System.out.println("Let's start the job again.");
        job.resume();
        System.out.println("Status is: " + job.getStatus().toString());
    }
    catch (Exception e)
    {
        System.out.println("Problem occoured:" + e.getMessage());
    }
}
```

As mentioned the API supports staging files, which means transferring files to a host before actually executing the job and transferring data files back as well. Since this is an essential function of the grid support I will show a simple example of how it is possible to transfer files.

Basically all you need is to add the following lines of code and everything will happen automatically:

```
jobspec.addStageInFile("/home/johndoe/input.txt",
    "/tmp/globus/input.txt");
jobspec.addStageOutFile("/tmp/globus/output-data.txt",
    "/home/johndoe/output-data.txt");
```

It is also possible to transfer files without doing it as part of a job submission. Basically you need to make a `GridDataManagement` object and then you can either use the `copy()` or the `put()` and `get()` functions. It is possible to work with either Java File objects, as in the following example, or work directly with URLs. The API also includes functions for cleaning up by deleting files no longer used.

```
GridDataManagement dm = new GridDataManagement(conn);

String localPath = "/home/kenneth/data/";
String gridPath = "gsiftp://dheghnom.dnsalias.com/storage/";

File inputFile = new File (localPath + "input-data.txt");
File outputFile = new File (localPath + "output-data.txt");
outputfile.createNewFile();

// Transfer the "input-data.xml" file.
try {
    // false == do not append.
    dm.put(inputFile, gridPath + "input-data.xml", false);
}
catch (SystemException e)
{
        // Error occured transferring file.
}

GridJob job = conn.submitJob(jobspec, "dheghnom.dnsalias.com", 8443);

// Simple polling
while (job.getStatus() == GridJobStatus.RUNNING)
{
    this.wait(300);
}

if (job.getStatus() == GridJobStatus.DONE)
{
        // Let's assume the job wrote the file output-data.xml
    try {
        dm.get(outputFile, gridPath + "output-data.xml", false);
    }
    catch
    {
        // Error occoured transferring file.
    }
}
```

Chapter 7

Integration Tests

Even though unit tests should cover some many rudimentary bugs, every system still has to be tested in a real environment in order to iron out as many bugs as possible. This is the so-called integration test.

In order to test the implemented system I need a running grid. Here at German Aerospace Center we have a minimal Globus-based grid system running that will allow me to test the basic functionality and on my home PC I have installed a Globus 4 environment as well.

In case everything works as expected that does not mean that the system is bug free, on the contraire, since it only allows me testing some parts. For instance, I cannot test if we can enforce a resource requirement of an ia86 processor if no one is available in our grid system, etc.

First I start out by defining tests that correspond to the major parts of the functionality implemented.

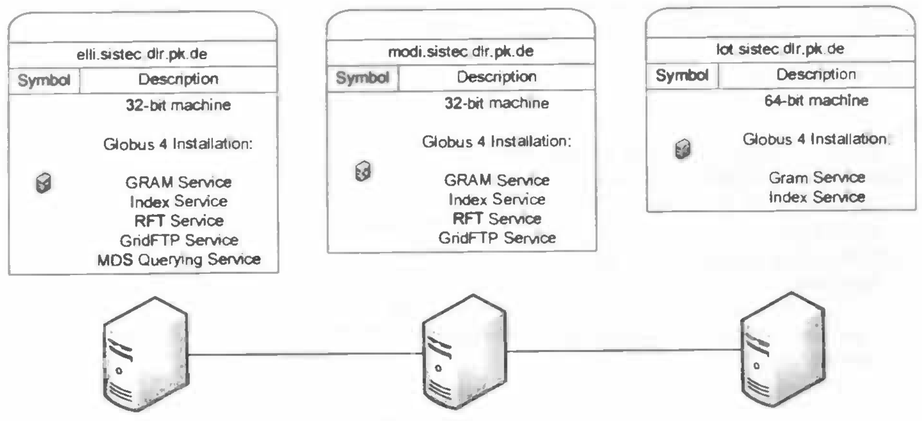
- Can we connect to the grid middleware system with our RCE certificates?
- Can we execute a basic job and redirect output?
- Can we run a job written in Java? (incl. file staging and setting more properties)[38]
- Can we suspend and resume a job?
- Can we enforce resource requirements (at submission time and with querying)?
- Can we transfer and delete files with the data management system?

The source code of the integration tests can be found in the appendix and serve as a good example source for using the Grid-aware SDK.

7.1 The Testing Environment

Grid support needs to be tested in a real grid environment so see if everything works as supposed, which regard to authorizing, transferring files etc. The finished system has been tested using a grid available here at German Aerospace Center consisting of 3 machines with Globus 4 installations, but running different services. These machines are Elli, Mode and Lot and their differences are listed below

[38]Running a Java based job, will require working staging of data files as well as setting of environment variables.



The testing environment at the German Aerospace Center.

Apart from testing at the German Aerospace Centre grid, a Globus installation has been made to one of our development machines (dheghnom.dsnalias.com), which made it possible to play a bit around without touching a machine in productivity.

7.2 The Tests

7.2.1 Test 1: Connection

Testing if we can connect to the underlying grid system with our RCE certificate is done the following way. I copy a legal credential file to our local user directory and construct a RCE `ProxyCertificate` from it. With this proxy-certificate I connect to the grid and print out an error in case we get an exception.

The important parts of the code for this are listed below:

```
public static ProxyCertificate getUserCertificate()
    throws GeneralSecurityException
{
    InputStream inputStream = new
        Object().getClass().getResourceAsStream("/usercert.pem");
    X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

    Date timestamp = new Date();
    timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

     return new ProxyCertificate(x509Cert, timestamp);
}

public static void main(String[] args) {
    ProxyCertificate cert;

    System.out.println("Running Integration Test #1");

    try {
        cert = getUserCertificate();
    } catch (GeneralSecurityException e) {
        System.out.println("Failed getting user certificate");
        return;
    }

    GridConnection conn;

    try {
        conn = Grid.authenticate(cert);
    } catch (SystemException e) {
        System.out.println("Couldn't not find the user certificate");
        return;
    } catch (UserException e) {
        System.out.println("Couldn't not authenticate with "
            + "underlying Grid middleware");
        System.out.println(e.getMessage());
        return;
    }
}
```

The above test runs with correct output and thus passes. In order to see what is happening, debug info has been added to the source code. In the future this debug info should be logged instead of printed out as now.

The output is:

```
Running integration test #1

We have authenticated with the underlying Grid middleware system.
TEST PASSED!
```

7.2.2 Test 2: Execution of Basic Grid Job: Linux Binary

In the above test we made a connection with the grid middleware system. How we want to see if we can describe a simple job consisting of executing the Linux binary "`echo`" on the grid and print out the text "Hello Grid World".

In order to see the output, we need to set the output stream to "`System.out`" and we are thus at the same time testing if output redirection works.

The central code for describing this job and submitting it is listed below.

```
GridJobSpecification spec = new GridJobSpecification();

spec.setResourceSpecification(resspec);
spec.setWorkingDirectory("/bin");
spec.setLocation("echo");
spec.addArgument("Hello Grid World!");
spec.setStderr(System.err);
spec.setStdout(System.out);

try {
    conn.submitJob(spec, "dheghnom.dnsalias.com", 8443);
} catch (UserException e) {
    System.out.println("Couldn't not authenticate with "
        + "underlying Grid middleware");
    System.out.println(e.getMessage());
    return;
} catch (SystemException e) {
    System.out.println("An error occured with out/input "
        + "redirection or the connection to the job "
        + "broker failed.");
    System.out.println(e.getMessage());
    return;
}
```

The code runs correctly with the following output:

```
Running integration test #2
Using provider: gt4 and jobmanager: fork
Using service contact: dheghnom.dnsalias.com
Setting directory to: /bin
Setting executable to: echo
Adding argument: Hello Grid World!
Setting stdout output to: (remote) /tmp/stdout-remote-1153111980508
Setting stderr output to: (remote) /tmp/stderr-remote-1153111980508

STATUS (Globus): Active
STATUS (Globus): Completed
FINISHED JOB
Stating out error output: file://dheghnom.dnsalias.com/tmp/stderr-remote-
1153111980508 = file://127.0.0.1/tmp/stderr-remote-1153111980508-transferred
Deleting: /tmp/stderr-remote-1153111980508
Deleting: /tmp/stderr-remote-1153111980508-transferred
Stating out output: file://dheghnom.dnsalias.com/tmp/stdout-remote-1153111980508 =
file://127.0.0.1/tmp/stdout-remote-1153111980508-transferred
Hello Grid World!          ◀ The output from the job is correctly printed
Deleting: /tmp/stdout-remote-1153111980508
Deleting: /tmp/stdout-remote-1153111980508-transferred
TEST PASSED!
```

7.2.3 Test 3: Execution of Java-based Grid Job

Executing a Java-based grid job is very similar to the above test case, with the difference that the "executable" is set to be "java" and the environment variables need to be set as well so that the Java class path is found. The first parameter is then set to be the Java class file containing the main class.

This is done the following way:

```
GridJobSpecification spec = new GridJobSpecification();
spec.setResourceSpecification(resspec);
spec.setWorkingDirectory("/usr/bin");
spec.setLocation("java");
spec.addArgument("StringReverser");
spec.addEnvironmentVariable("CLASSPATH",
    "$CLASSPATH:/home/kenneth/");
spec.setStderr(System.err);
spec.setStdout(System.out);
```

Additionally we need to stage in the compiled `.class` file (`StringReverser.class`). Since we already require staging to work, we test a more interesting Java application that reads in a string from `input-data.txt`

```
abcdefghijklmnopqrstuvwxyz
```

And reverses the string and writes it back to `output-data.txt`. This requires us to stage in the `input-data.txt` and stage out the `output-data.txt` file. We do this the following way:

```
        spec.addStageInFile("/home/kenneth/workspace/" +
            "RCEIntegrationTests/StringReverser.class",
            "/home/kenneth/StringReverser.class");
        spec.addStageInFile("/home/kenneth/workspace/" +
            "RCEIntegrationTests/input-data.txt",
            "/home/kenneth/input-data.txt");
        spec.addStageOutFile("/home/kenneth/output-data.txt",
            "/home/kenneth/workspace/" +
            "RCEIntegrationTests/output-data.txt");
```

The basic code used to reverse the string is listed below. The full file can be found in the appendix.

```
private static String reverse(String str) {
    StringBuilder sb = new StringBuilder(str);

    return sb.reverse().toString();
}

public static void main(String[] args) {
    FileReader reader = null;
    FileWriter writer = null;

    try {
        reader = new FileReader("/home/kenneth/input-data.txt");
    } catch (FileNotFoundException e) {
        System.out.println("Cannot find /home/kenneth/input-data.txt");
    }

    BufferedReader in = new BufferedReader(reader);

    String str = null;

    try {
        str = in.readLine();
    } catch (IOException e) {
        System.out.println("Couldn't read the file");
    }

    String revStr = reverse(str);

    File output = new File("/home/kenneth/output-data.txt");

    try {
        output.createNewFile();

        writer = new FileWriter(output);

        writer.write(revStr, 0, revStr.length());
        writer.flush();
        writer.close();
    } catch (IOException e) {
        System.out.println("Couldn't create "
            + "/home/kenneth/output-data.txt");
    }
}
```

The test passes and the output is:

```
Running integration test #3
Using provider: gt4 and jobmanager: fork
Using service contact: dheghnom.dnsalias.com
Setting directory to: /usr/bin
Setting executable to: java
Adding argument: StringReverser
Adding environment variable: CLASSPATH=$CLASSPATH:/home/kenneth/
Setting stdout output to: (remote) /tmp/stdout-remote-1153112083664
Setting stderr output to: (remote) /tmp/stderr-remote-1153112083664

STARTING STAGING IN: 2 files
Adding file to stage in:
file://127.0.0.1/home/kenneth/workspace/RCEIntegrationTests/input-data.txt =
file://dheghnom.dnsalias.com/home/kenneth/input-data.txt
Adding file to stage in:
file://127.0.0.1/home/kenneth/workspace/RCEIntegrationTests/StringReverser.class =
file://dheghnom.dnsalias.com/home/kenneth/StringReverser.class
STATUS (Globus): Active
STATUS (Globus): Completed
FINISHED JOB
STARTING STAGING OUT: 1 files.
Adding file to stage out: file://127.0.0.1/home/kenneth/output-data.txt =
file://dheghnom.dnsalias.com/home/kenneth/workspace/RCEIntegrationTests/output-
data.txt
Stating out error output: file://dheghnom.dnsalias.com/tmp/stderr-remote-
1153112083664 = file://127.0.0.1/tmp/stderr-remote-1153112083664-transferred
Deleting: /tmp/stderr-remote-1153112083664
Deleting: /tmp/stderr-remote-1153112083664-transferred
Stating out output: file://dheghnom.dnsalias.com/tmp/stdout-remote-1153112083664 =
file://127.0.0.1/tmp/stdout-remote-1153112083664-transferred
Deleting: /tmp/stdout-remote-1153112083664
Deleting: /tmp/stdout-remote-1153112083664-transferred
TEST PASSED!

$ cat /home/kenneth/output-data.txt      ◀ We look at the output data file
zyxwvutsrqponmlkjihgfedcba
$
```

7.2.4 Test 4: Suspending and resuming

In order to test if we can suspend and resume we write a simple counter in java:

```java
public class Counter {
    public static void main(String[] args) throws InterruptedException {
        int count = 0;

        while (count < 10) {
            count++;
            Thread.sleep(200);
            System.out.println(count);
        }
    }
}
```

This above grid job will print out 10 number from [1...10]. If it suspends and resumes properly it will stop printing out numbers when suspended and resume when resumed.

```java
try {
    job.suspend();
    job.resume();
} catch (SystemException e) {
    e.printStackTrace();
} catch (UserException e) {
    e.printStackTrace();
}

   Do not exit Java client before the grid job has finished
   This way we will be sure to see all output.
while (job.getStatus() != GridJobStatus.DONE)
{
    ;
}
```

Getting suspend and resume working required a few changed to the code and to the Globus installation. Basically it requires using a different job manager that supports suspends and resumes, as for instance Condor[39].

The output of the test was:

```
Running integration test #4
Using provider: gt4 and jobmanager: condor
Using service contact: dheghnom.dnsalias.com
Setting directory to: /usr/bin
Setting executable to: java
Adding argument: Counter
Adding environment variable: CLASSPATH=$CLASSPATH:/home/kenneth/
Setting stdout output to: (remote) /tmp/stdout-remote-1152973960650
Setting stderr output to: (remote) /tmp/stderr-remote-1152973960650

STARTING STAGING IN: 1 files
Adding file to stage in:
file://127.0.0.1/home/kenneth/workspace/RCEIntegrationTests/Counter.class =
file://dheghnom.dnsalias.com/home/kenneth/Counter.class
STATUS (Globus): Active
SUSPEND ISSUED                ◄ Suspend is issued... and the output this far is printed
STATUS (Globus): Suspended
Stating out error output: file://dheghnom.dnsalias.com/tmp/stderr-remote-
1152974044697 = file://127.0.0.1/tmp/stderr-remote-1152974044697-transferred
Deleting: /tmp/stderr-remote- 1152974044697
Deleting: /tmp/stderr-remote-1152974044697-transferred
Stating out output: file://dheghnom.dnsalias.com/tmp/stdout-remote-1152974044697 =
file://127.0.0.1/tmp/stdout-remote-1152974044697-transferred
1
2
3
4
Deleting: /tmp/stdout-remote-1152974044697
Deleting: /tmp/stdout-remote- 1152974044697-transferred
RESUME ISSUED                 ◄ Resume is issued...
STATUS (Globus): Resumed
STATUS (Globus): Completed
FINISHED JOB
Stating out error output: file://dheghnom.dnsalias.com/tmp/stderr-remote-
1152974044697 = file://127.0.0.1/tmp/stderr-remote-1152974044697-transferred
Deleting: /tmp/stderr-remote- 1152974044697
Deleting: /tmp/stderr-remote-1152974044697-transferred
Stating out output: file://dheghnom.dnsalias.com/tmp/stdout-remote-1152974044697 =
file://127.0.0.1/tmp/stdout-remote-1152974044697-transferred
5
6
```

[39] http://www.cs.wisc.edu/condor/

```
7
8
9
10
Deleting: /tmp/stdout-remote-1152974044697
Deleting: /tmp/stdout-remote-1152974044697-transferred
TEST PASSED!
```

7.2.5 Test 5: Enforcing Resource Requirements

The developed SDK allows for checking resource requirements, so that is important to test as well. First we need to set a requirement, and we decide to require at least one CPU:

```
GridResourceSpecification resspec = new GridResourceSpecification();

try {
    resspec.setNoOfProcessors(1);
} catch (UserException e1) {
     e1.printStackTrace();
}
```

Now that a requirement has been set, we need to do the actual querying.

```
boolean passed = conn.queryResources(resspec, "dheghnom.dnsalias.com",
    8443);

if (passed) {
    System.out.println("We passed the resource requirements");
} else {
    System.out.println("We didn't pass the resource requirements");
    return;
}
```

The output of the test is:

```
Running integration test #5
Resource requirement: We require 1 CPU(s).        ◄ Output from querying

We passed the resource requirements               ◄ We do indeed have one CPU!
Using provider: gt4 and jobmanager: condor
Using service contact: dheghnom.dnsalias.com
Setting directory to: /bin
Setting executable to: echo
Adding argument: Hello Grid World!

Resource requirement: We require 1 CPU(s).        ◄ Output from job submission

Setting stdout output to: (remote) /tmp/stdout-remote-1153112158951
Setting stderr output to: (remote) /tmp/stderr-remote-1153112158951
STATUS: Active
STATUS: Completed
FINISHED JOB
Stating out error output: file://dheghnom.dnsalias.com/tmp/stderr-remote-
1153112158951 = file://127.0.0.1/tmp/stderr-remote-1153112158951-transferred
Deleting: /tmp/stderr-remote-1153112158951
Deleting: /tmp/stderr-remote-1153112158951-transferred
Stating out output: file://dheghnom.dnsalias.com/tmp/stdout-remote-1153112158951 =
file://127.0.0.1/tmp/stdout-remote-1153112158951-transferred
Hello Grid World!
```

```
Deleting: /tmp/stdout-remote-1153112158951
Deleting: /tmp/stdout-remote-1153112158951-transferred
TEST PASSED!
```

Correctly we passed the test, but we cannot know if it really works before we have tested something that is supposed to fail. For this reason we change the number of required CPUs to 2. Correctly, the test fails as can be seen below:

```
Running integration test #5
Resource requirement: We require 2 CPU(s).    ◀ Output from querying

We didn't pass the resource requirements      ◀ Correctly, we get an error
```

7.2.6 Test 6: Transfers with the Data Management System

In the above test we transferred data files along with the job, but the grid support also supports transferring files independently of job execution by use of the Data Management facility.

In order to test the functionality we create a file with Java and we transfer it to a grid node using the `put()` method. This method makes it possible to transfer a Java File object. Afterward we list the content of the directory.

This is done by the following code, here listed without error handling:

```
    create local file
    write to file

String fileName = "/home/kenneth/fileToPut.txt";

System.out.println("Creating with " + fileName
    + " with the content \"hello grid world\"");

File fileToPut = new File(fileName);
BufferedWriter bw;

try {
    fileToPut.createNewFile();
    bw = new BufferedWriter(new FileWriter(fileToPut));

    String[] words =  {"hello", "grid", "world" };

    for (String word: words) {
        bw.write(word + " ");
    }
    bw.flush();
} catch (IOException e) {
    e.printStackTrace();
}

    copy (put) file to grid node
    list directory

String gsiport = "5678";
String gsiserver = "dheghnom.dnsalias.com";
String address = "gsiftp://" + gsiserver + ":" + gsiport;

System.out.println("We copy the file to the grid node "
    + "and list the directory");
String[] files = null;

try {
    management.put(fileToPut, address + "/home/kenneth/grid/greeting.txt",
        false);

    files = management.listDirectory(address + "/home/kenneth/grid/");
} catch (UserException e) {
    e.printStackTrace();
} catch (SystemException e) {
    e.printStackTrace();
}

System.out.println("The directory contains the "
    + "following file(s):");

for (String file: files) {
    System.out.print(file + " ");
}
System.out.println();
```

We also want to make sure that we can receive files, so we create a File object used to receive the file and we print the content with use of standard Java methods:

```
   create new local file
   copy (get) file from grid node
   read   print contents of the file

fileName = "/home/kenneth/fileToGet.txt";

System.out.println("Creating the file " + fileName
    + " which will be used to receive contents");

File fileToGet = new File(fileName);

try {
    fileToGet.createNewFile();
} catch (IOException e) {
    e.printStackTrace();
}

try {
    management.get(fileToGet, address +
         "/home/kenneth/grid/greeting.txt", false);
} catch (UserException e) {
    e.printStackTrace();
} catch (SystemException e) {
    e.printStackTrace();
}

BufferedReader br;

System.out.println("Reading and printing out the content of " + fileToGet);

try {
    br = new BufferedReader(new FileReader(fileToGet));
    String line;

    while (true) {
        if ((line = br.readLine()) == null) {
            break;
        }
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

The output of the test is:

```
Running integration test #6

Creating with /home/kenneth/fileToPut.txt with the content "hello grid world"
We copy the file to the grid node and list the directory
Copying file:///home/kenneth/fileToPut.txt to
    file://dheghnom.dnsalias.com/home/kenneth/grid/greeting.txt
Listening the directory /home/kenneth/grid
The directory contains the following file(s):
greeting.txt            ◀ File has been copied to the grid node
Creating the file /home/kenneth/fileToGet.txt which will be used to receive
    contents
Copying file://dheghnom.dnsalias.com/home/kenneth/grid/greeting.txt to
    file:///home/kenneth/fileToGet.txt
Reading and printing out the content of /home/kenneth/fileToGet.txt
hello grid world        ◀ We received the file with the contents we wrote to it.
TEST PASSED!
```

Chapter 8

Evaluation

After having finished the implementation of the grid-aware software development kit, which adds grid support to the method SDK in RCE, I will evaluate the result, starting with looking at the software engineering strategy.

As the description of the software engineering strategies shows, a lot of effort at German Aerospace Center has been put into making the RCE and SESIS project use these strategies to the outmost, in order to avoid the most common pitfalls and not to end up with an unmaintainable project.

It can be hard measuring the success of using such strategies as many of the benefits first show themselves after a long time period. For instance, it is difficult knowing if the time saved on for instance maintenance makes up for the extra time spent on design. Doing software engineering right takes time and some will argue that it takes the fun away from programming.

During the implementation I recognized this; a lot of time is spend on writing unit tests, making sure the code style is following the style guide (if not, it does not compile due to the use of Checkclipse) and making sure that I am following the design that I have made in the design phase. Programming becomes more manual work, but it is encouraging to see how much works right away and how many – hard to find – bugs have been caught by the unit tests. Using the software engineering strategies outlined earlier in the document, has definitely been a win in my opinion.

The implementation has been done using the CoG Toolkit and it has been moderate success. The CoG API is clear and easy to use and it provides the functionality needed. CoG has also proved stable and reasonable bug-free, which is also the experience other German Aerospace Center employees have had using CoG. Nothing is perfect, and even though CoG is fully documented, the documentation is sparse in some areas; for example explaining that a function returns an integer representing a status value, without explaining what these values can be and what meaning they have. Most of these problems have been solved by relying on the source code as documentation or by finding examples on the internet.

What I have implemented during the thesis period is a software development kit that adds grid support to the RCE platform, so that it is possible to write RCE applications consisting of one of more grid jobs. The implementation of the support makes it possible to:

- Transparently access a grid middleware system
- Specify resources and query for them
- Specify jobs with all requirements such as file staging
- Submit jobs and suspend and restart them at need
- Deal with input and output of grid jobs
- Transfer data files from grid node to grid node

This covers most of the needs of the RCE users and I am thus satisfied with what has been accomplished.

After having worked with grid computing for now half a year it is clear to me that grid computing is a huge field within computer science that you don't gasp before having spent much time working with it.

Grid middleware systems are complex software systems consisting of various services accessible either though an abstraction API or thought a web service interface. Understanding the services and web service architecture requires effort even though the final solution seems very simple.

It has also come to my attention that grid computing is much more than what I initially expected it to be and it will surely have my attention for the years to come.

Chapter 9

Future Work

Software projects are never really finished, as there are always things there can be improved and there are always new surrounding and technologies to integrate with. It is no different with the grid support added to RCE.

During this project a lot of technology study has been made in order to find out what exactly Grid Computing is, how it fits into the RCE system and what software that is future-proof and well-functioning to be used in the RCE project. Additionally, a design has been made that makes it possible to use grid resources from RCE using one of two grid middleware systems: UNICORE or Globus. As only Globus support has actually been implemented, UNICORE is a candidate for future work.

Another obvious candidate for future work would be testing the implemented grid-aware SDK in more grid configurations to iron out the remaining bugs and then to review the code once again and clean it up as the integration testing has revealed bugs that have resulted in code changes.

Regarding file transfers, support has been implemented for the file, HTTP and GridFTP protocols, but Globus additionally supports the RFT (Reliable File Transfer) protocol. An option for future work would be adding support for this particular protocol.

Also, only one of the suggested areas of grid support has been implemented, and as such it should be possible to add even tighter grid support, though it is questionable how much this will bring the RCE project as the SDK grid support generally covers the use-cases presented by the SESIS customers.

The extension methods written for the RCE system by third parties are often very specific and written to accomplish a specific task. Because of this nature they are either written to take advantage of a computational grid or not. This makes support like what ProActive offers less attractive for RCE, which is more appropriate for applications that want to take advantage of a scavenging grid if available, which is not the case here.

Chapter 10

Conclusion

The purpose of this thesis was analyzing how and if the RCE platform can take advantage of grid resources. During the thesis I showed various areas where grid technology could complement RCE and presented these in 3 groups of options.

The option was chosen to develop a Grid-aware Software Development Kit that would integrate with the RCE system, by for instance reusing the RCE certificated. Such a SDK has been designed and in such a way that it should be possible to develop new back ends. As part of the thesis a backend for the Globus middleware system has been developed.

I believe the right option was chosen, as the grid-aware SDK makes it possible for third-party developers taking advantage of grid resources and grid technology when developing extension methods. Extension methods are often written when products like SESIS are deployed due to the fact that SESIS is not an out-of-the-box magical solution to ship development, but instead a product that when integrated with the current work chain and software stack can help the companies ease their workflow.

The extension methods are often used for interacting with other software tools or for performing specific calculations; calculations that might be able to take advantage of grid technology, due to the fact that the grid provides the ability to deal with large data sets, as well as solve large-scale computational problems that are too complex for one machine to handle.

I believe that the developed Grid-aware SDK makes this possible by for instance letting these extension methods consist of several grid jobs. All that should be needed to do this is included in the grid-aware SDK.

Coming back to the question whether is was possible to take advantage of grid resources in RCE; we can now conclude that this is indeed the case, which the developed extensions (the Grid-aware SDK) to RCE are a living proof of.

Chapter 11

Acknowledgements

I would like to thanks the people at SISTEC, DLR (German Aerospace Centre) for housing me and letting me experience how software engineering can be put to use in larger projects. It has been a great experience and I'm thankful. I would like to thank Andreas Schreiber and Thijs Metsch for commenting on this master thesis.

I would also like to thank my advisor Sietse Achterop at the University of Groningen, the Netherlands for guidance and suggesting on how to improve my thesis.

Special thanks go to Emma Greenley for reading thought this technical paper and improving my English and to Kristen Nielsen for reading my thesis twice and helping me improving the structure.

Chapter 12

Appendix

12.1 References

[Brooke]	John Brooke, Donal Fellows, Kevin Garwood and Carole Goble, "*Semantic matching of Grid Resource Descriptions*". Web Reference: http://www.Grid-interoperability.org/semres.pdf
[Brooke-2]	John M. Brooke, Donal Fellows and Jon MacLaren, "*Interoperability of Resource Description across Grid Domain Boundaries*", Web Reference: http://www.uniGrids.org/papers/Interoperability%20of%20resource%20descriptions.pdf
[Bihler2005]	Pascal Bihler, 2005. "*Come together, right now!*". Web Reference: http://www.bi-on.de/mixed/pdf/Grids_text.pdf
[CC]	ThoughtWorks, "*CruiseControl, continuous integration toolkit*". Web Reference: http://cruisecontrol.sourceforge.net
[Checklipse]	Benjamin Livshits, 2005, "Checkclipse: Finding Bugs in Eclipse Code using Eclipse". Web Reference: http://suif.stanford.edu/~livshits/work/checklipse/checklipse.html
[CoG]	The Globus Alliance, 2006, "*CoG: Java Commodity Grid Kit*", Web Reference: http://www.globus.org/cog/
[CVS]	The CVS Team, "*Concurrent Versions System*", Web Reference: http://www.nongnu.org/cvs/ Wikimedia Foundation, "*CVS Wikipedia Site*", Web Reference: http://en.wikipedia.org/wiki/Concurrent_Versions_System
[D-Grid]	Forschungszentrum Karlsruhe, "*D-Grid Initiative*". Web Reference: http://www.d-Grid.de/
[Eclipse]	The Eclipse Foundation, 2006, "*Eclipse.org home*". Web Reference: http://www.eclipse.org
[Globus]	The Globus Alliance, 2006. "*The Globus Alliance and the Globus Grid Toolkit*". Web Reference: http://www.globus.org/
[GPE]	Rob van Nieuwpoort, University of Amsterdam. "*Getting started with the Grid Application Toolkit*", Web Reference: http://www.gridlab.org/WorkPackages/wp-1/Doc/JavaGAT-tutorial.pdf
[GPE]	Ralf Ratering, Intel, 2004. "*Grid Programming Environment (GPE)*", Web Reference: https://www.Gridlab.org/WorkPackages/wp-1/Presentations/Edinburgh-07-2004/slides_-_hrabri.pdf

Ralf Ratering, Intel, 2004. "*UNICORE, what it is and where it goes...*". Web Reference: http://www.Grid.lrz.de/en/talk/Gridtag04/rratering.pdf

Thomas Kentemich, Intel PDSD, 2005. "*UniGridS and GPE: A Client Framework for Interoperability*", Web Reference: http://www.summit.unicore.org/2005/Thomas_Kentemich.pdf

Ralf Ratering, Intel, 2005. "*Grid Programming Environment (GPE) Concepts*", Web Reference: http://prdownloads.sourceforge.net/unicore/GPE-Concepts.pdf?download

---

[IWAY] I. Foster and J. Geisler and W. Nickless and W. Smith and S. Tuecke, 1997. "*Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment*", Web Reference: ftp://ftp.globus.org/pub/globus/papers/isoft.pdf

---

[JUnit] Object Mentor, Incorporated. "*JUnit, Testing Resources for Extreme Programming*", Web Reference: http://www.junit.org/

[Mantis] The Mantis Team, "Mantis Bug Tracking System". Web Reference: http://www.mantisbt.org/

[Maven] Apache Software Foundation, "*Apache Maven Project*", Web Reference: http://maven.apache.org/

[OGSI] ANL, IBM, USC/ISI, Fujitsu Labs, NASA, 2003, "*Open Grid Services Infrastructure (OGSI) Version 1.0*". Web Reference: http://xml.coverpages.org/OGSI-SpecificationV110.pdf

---

[OSGi] OSGi Alliance (formerly known as the Open Services Gateway initiative), "*The OSGi Service Platform - Dynamic services for networked devices*". Web Reference: http://www.osgi.org/

Wikimedia Foundation, "*OSGi Wikipedia Site*", Web Reference: http://en.wikipedia.org/wiki/OSGi

---

[ProgTut] Borja Sotomayor, University of Chicago. "*The Globus Toolkit 4 Programmer's Tutorial*". Web Reference: http://gdp.globus.org/gt4-tutorial/singlehtml/progtutorial_0.2.1.html

---

[Psysio] Ian Foster, Carl Kesselman Jeffrey M. Nick Steven Tuecke. "*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*". Web Reference: http://www.globus.org/alliance/publications/papers/ogsa.pdf

---

[SESIS] Deutsches Zentrum für Luft- und Raumfahrt, SCAI Fraunhofer. "*SESIS – Integriertes Schiffsentwurfs – und Simulationssystem*". Web Reference: http://www.scai.fraunhofer.de/sesis.html

Thomas Brandes, Katja Christiansen, Eric Esins, Jürgen Klein, Ottmar Krämer-Fuhrmann, Thijs Metsch, Dirk Rossow, Andreas Schreiber and Sandre Schrödter, 2005. „*System-Design für ein schiffbauliches Entwurfs- und Simulationssystem*". Internal Document.

Thijs Metsch, Andreas Schreiber, 2006. "*RCE Development Standard*". Internal Document.

[Subclipse]	The Subclipse Team, "*Subclipse: A Subversion Eclipse Plugin*", Web Reference: http://subclipse.tigris.org/
[Subversion]	The Subversion developers, "*Subversion Website*", Web Reference: http://subversion.tigris.org/
	Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, 2005-2006, "*Version Control with Subversion*", O'Reilly Media, Web Reference: http://www.oreilly.com/catalog/0596004486/ and free online read: http://svnbook.red-bean.com/
	Wikimedia Foundation, "*Subversion (software) Wikipedia Site*", Web Reference: http://en.wikipedia.org/wiki/Subversion_%28software%29
[UNICORE]	UNICORE Forum e.V. 2006. "*UNICORE Forum e.V. and the UNICORE Grid Toolkit*", Web Reference: http://www.unicore.org/
[UniGridS]	Forschungszentrum Jülich GmbH, CINECA, Fujitsu Labaratories of Europe, University of Warsaw, Intel GmbH, The University of Manchester, T-Systems SfR, 2005. "*Uniform Interface to Grid Services*", Web Reference: http://www.uniGrids.org/
[WSDL]	W3C, Microsoft, IBM Research, "*Web Services Description Language (WSDL) 1.1*", Web Reference: http://www.w3.org/TR/wsdl
[WSRF]	The Globus Alliance 2005. "*The WS-Resource Framework*", Web Reference: http://www.globus.org/wsrf/
	OASIS, IBM, Oracle, Computer Associates, 1 April 2006, "*Web Services Resource 1.2 (WS-Resource) OASIS Standard*", Web Reference: http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

12.2 Terminology

A thesis of technical art will always contain terminology used in the given field of the subject; terminology that can be quite hard to gasp for most people not daily working in the area. In order to make this thesis more easily readable a terminology list has been compiled. It consists of words that the authors and the reviewers have had difficulty understanding.

API

An acronym for Application Programming Interface. The interface specifies calling conventions by which an application program uses a service/application such as RCE. An API is defined at source code level and provides a level of abstraction between the user application/application extension and the service/application to ensure the portability of the code.

(Computer) Bug

A bug in computer terminology is an error, flaw, mistake, failure or fault in a program that prevents it from working correctly or produces an incorrect result.

Bundle

The name of a component in a particular (OSGi) implementation of a component system.

Component

An object adhering to a component architecture. In a component-based architecture, the components of a system have generic interfaces through which they advertise their functionalities, enabling the dynamic loading of the components.

Dead code

In programming, dead code consists of code blocks or methods (routines) that are never accessed. This can either be because calls to them have been removed or because they are guarded by a control structure that transfers the control somewhere else.

Executable

A file that contains a program that is capable of being executed or run on a computer.

Garbage Collector

A virtual machine process employed to collect and dispose data objects that can no longer be used and makes their memory available for reuse.

GUI

An acronym for Graphical User Interface; a term that refers to a software front-end that provides an attractive and easy-to-use interface between a computer user and an application.

High throughput (computing)

Throughput is a measure of the amount of data transferred in a specific amount of time, usually expressed as bits per second (bps).

(Extension) Method

Extension methods are components that extent RCE with for instance simulation or calculation methods. These methods can be pure data providers, real applications or wrappers of other external applications and legacy source code.

NFS

An acronym for Network File System, which is a protocol suite developed and licensed by Sun Microsystems that allows different makes of computers running different operating systems to share files and disk storage. In many ways it is similar to SMB known in the Windows world.

(Grid) node

A device attached to a network, which in the case of a Grid node is a computer running Grid middleware. A node uses the network as a means of communication and has an address on the network.

Plug-in

A plug-in (or plugin) is a program that plugs into an application in order to extend the functionality. Plug-ins are normally dynamically loaded on startup.

RCE

RCE is a base system that offers a distributed system for accessing and managing data, as well as for accessing and using so-called extension methods that extent RCE with for instance simulation or calculation methods. RCE is build up around a virtual organization structure which means that a distributed RCE installation, given the appropriate rights, can be used my multiply organizations to cooperate, share data, as well as use each others methods. RCE is in itself not a full product and only specialized versions of it are put on the market.

Remote system

A system residing on another computer than the one the user is using.

SDK

An acronym for Software Development Kit. A collection of programming tools, utilities, documentation, and libraries that allows software developers to create products to run on a particular platform or to work with an API.

SESIS

SESIS is a conceptual design and simulation system for the early design phases of ship development. The system empowers the engineers to perform complex collaborative simulations between the shipyards and suppliers over the internet, by levering the features offered by RCE. SESIS is in fact, a specialized version of RCE supplied designed with ship design in mind and as such sold with extra standard methods usable when designing ships.

Server-side, client-side

A server-side program is a program that reside on the server and that a user can interact with through the network without actually downloading the program. A client-side program, is a program that runs on the computer in use.

Software Portability

A measure of system independence, in the sense that truly portable programs can be moved to a new system by recompiling without having to make any changes to the source code.

Stateful, stateless

The adjectives stateful and stateless describe whether a computer or computer program can remember one or more preceding events in a given sequence of interactions.

Virtual Organization

A network of companies, suppliers, customers, or employees, linked by information and communications technologies, with the purpose of delivering a service or product.

12.3 Java Code Conventions for the RCE and SESIS projects

In the following we describe the Java code conventions used in the RCE and SESIS projects. By using the Checkclipse plugin for Eclipse and the configuration file in the RCE repository these coding conventions will actually be enforced within Eclipse. The coding conventions are also checked upon check-in to the repository and if not followed the check-in is rejected.

These code conventions are from the internal German Aerospace Center document entitled "*Code Conventions for the Java Programming Language*", written by Thijs Metsch and they are derived from SESIS team meetings.

12.3.1 File Suffixes

File Type	Suffix
.java	Java source
.class	Java byte code

12.3.2 Organization of the Content of a Source Code File

1. File header, describing class or interface/copyright/etc.
2. Package and Import statements
3. Class or interface statement
4. Optional class or interface comment
5. Static variables
    a. Public static variables
    b. Protected static variables
    c. Package level static variables
    d. Private static variables
6. Instance variables
    a. Public instance variables
    b. Protected instance variables
    c. Package level instance variables
    d. Private instance variables
7. Constructors
8. Methods (functionally grouped, no grouping necessary for scope or accessibility)

12.3.3 Indentation

- Indent 4 spaces
- Don't indent top level classes/interfaces
- Do indent members

When an expression will not fit on a single line (max 120), break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

12.3.4 Comments

- Don't explain what's obvious from the code
- Block comments must be preceded by a blank line
- Don't use single-Line comments for multiple lines
- Use Javadoc standard comments!

```
// Single Line comment.

/*
 * Here is a block comment.
 */

/**
 * Here is a Javadoc comment.
 */
```

12.3.5 Declarations

- One declaration per line so that comments are easy to add
- Use one space or one tab between type and identifier
- When possible, initialize a local variable with the declaration
- Put declarations at the beginning of the block (exception: for loops)
- Try not to reuse variable names from an outer block

12.3.6 Statements

- Each line should contain at most one statement.
- Compound Statements are statements that contain lists of statements enclosed in braces "{ statements }".
  - The enclosed statements should be indented one more level
  - The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
  - Braces are used around all statements, even single statements, when they are part of a control structure

A return statement with a value should not use parentheses

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

The if-else class of statements should have the following form:

```
if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

Loop statements should have the following form:

```
for (initialization; condition; update) {
    statements;
}


while (condition) {
    statements;
}
do {
    statements;
} while (condition);
```

A switch statement should have the following form:

```
switch (condition) {
case ABC:
    statements;
    /* falls through */
case DEF:
    statements;
    break;
default:
    statements;
    break;
}
```

A try-catch statement should have the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

12.3.7 White Space

- Use blank lines between related parts of the code
- Use two blank lines

        - between methods
        - between local variables and the first statement
        - before a comment (exception: end of line comment)
        - to improve readability: between logical sections in a method
- Use blank spaces:
    - Between a keyword and a parenthesis
    - After commas in argument lists
    - Before and after binary operators
    - Between expressions in a for statement
    - After a cast

12.3.8 Naming Conventions

Rules for Naming	**Example**
Package Names start with your domain in reversed order (lowercase)	`de.rcenvironment.rce`

Class/Interface names are nouns, mixed case, first letter capitalized	`class Raster, ImageSprite`
	`interface RasterDelegate, Storing`

Methods: verbs, mixed case, first letter lowercase	`run(), runFast(), getBackground()`

Variables: Mixed case, first letter lowercase. Do not start with underscore or $. Member-Variables will start with my. Avoid one letter variable names except for temporary variables
• Use c, d , and e for temporary character variables
• Use i, j, k, m and n for temporary integer variables
	`int   i;`
	`char  c;`
	`float myWidth;`

Constants: all uppercase, separate words	`static final int GET_THE_CPU = 1;`

Exception: Same as variable or use e	`Exception e`
	`SAXException e`
	`SAXException saxException`

12.3.9 Sample Code

```
 *
 * Bean.java
 *
 * Class description goes here.
 *
 * Created: [illegible] (mst: [illegible].metsch@dlr.de)
 * Changed:
 *
 * Copyright (C) [illegible] DLR SC, FHG SCAI, Germany
 *
 * All rights reserved.
 *
 * http:  www.rcenviroment.de
 */
package java.blah;

import java.blah.blahdy.BlahBlah;

 /**
  * Class description goes here.
  *
  * @version        $Revision 108
  * @author         Thijs Metsch
  */
public class Blah extends SomeClass {
      * A class implementation comment. */

      /** classVar documentation comment */
    public static int classVar;

     /** instanceVar documentation comment */
    private Object[] instanceVar;

     /**
      * ...constructor Blah documentation comment...
      */
    public Blah() {
        ... ...implementation goes here...
    }

     /**
      * ...method doSomethingElse documentation
      * comment...
      * @param someParam description
      */
    public void doSomethingElse(Object someParam)
            ...implementation goes here...
    }
 }
```

12.4 Source Code of the Integration Tests

Since the source code of the integration tests solves as good examples of how to use the public API of the Grid-aware SDK, they are listed in this appendix. Explanation to parts of the code can be found in *chapter 7*.

IntegrationTest_1.java:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;
import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.rce.sdk.grid.Grid;
import de.rcenvironment.rce.sdk.grid.GridConnection;

 /**
  * Integration test 1: Connection.
  *
  * @version $LastChangedRevision$
  * @author Kenneth Rohde Christiansen
  */
public class IntegrationTest_1 {

    /**
     * Period of validity.
     */
    private static final long PERIOD_OF_VALIDITY = 99999990;

    /**
     * The path to the users' home directory.
     */
    private static final String HOME_PATH =
        System.getProperty("user.home");

    /**
     * The system specific file separator.
     */
    private static final String FILE_SEPARATOR =
       System.getProperty("file.separator");

    /**
     * Gets user certificate by reading it from the harddrive.
     *
     * @return an RCE ProxyCertificate
     * @throws GeneralSecurityException
     */
    public static ProxyCertificate getUserCertificate() throws
        GeneralSecurityException {
        InputStream inputStream = null;
        try {
            inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
                + ".globus" + FILE_SEPARATOR + "usercert.pem");
        } catch (FileNotFoundException e) {
            System.out.println("User Certificate not found: " + HOME_PATH
                + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
                + "usercert.pem");
        }

        X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

        Date timestamp = new Date();
        timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

        return new ProxyCertificate(x509Cert, timestamp);
    }

    /**
     * Main function.
     *
     * @param args
     */
    public static void main(String[] args) {
        ProxyCertificate cert;

        System.out.println("Running integration test #1");

        try {
            cert = getUserCertificate();
        } catch (GeneralSecurityException e) {
            System.out.println("Failed to get user certificate");
            return;
        }

        GridConnection conn;

        try {
            conn = Grid.authenticate(cert);
        } catch (SystemException e) {
            System.out.println("Couldn't not find the user certificate");
            return;
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        }

        System.out.println("We have authenticated with the underlying "
            + "Grid middleware system.");
        System.out.println("TEST PASSED!");
    }
}
```

IntegrationTest_2.java:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.globus.gsi.CertUtil;
import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.rce.sdk.grid.Grid;
import de.rcenvironment.rce.sdk.grid.GridConnection;
import de.rcenvironment.rce.sdk.grid.GridJob;
import de.rcenvironment.rce.sdk.grid.GridJobStatus;
import de.rcenvironment.rce.sdk.grid.GridJobSpecification;
import de.rcenvironment.rce.sdk.grid.GridResourceSpecification;

/**
 * Integration test 1: Execution of Basic Grid Job: Linux Binary.
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class IntegrationTest_2 {

    /**
     * Period of validity.
     */
    private static final long PERIOD_OF_VALIDITY = 99999990;

    /**
     * The path to the users' home directory.
     */
    private static final String HOME_PATH =
        System.getProperty("user.home");

    /**
     * The system specific file separator.
     */
    private static final String FILE_SEPARATOR =
       System.getProperty("file.separator");

    /**
     * Gets user certificate by reading it from the hardrive.
     *
     * @return an RCE ProxyCertificate
     * @throws GeneralSecurityException
     */
    public static ProxyCertificate getUserCertificate() throws
        GeneralSecurityException {
        InputStream inputStream = null;
        try {
            inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
                + ".globus" + FILE_SEPARATOR + "usercert.pem");
        } catch (FileNotFoundException e) {
            System.out.println("User Certificate not found: " + HOME_PATH
                + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
                + "usercert.pem");
        }

        X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

        Date timestamp = new Date();
        timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

        return new ProxyCertificate(x509Cert, timestamp);
    }
```

```
/**
 * Main function.
 *
 * @param args
 */
public static void main(String[] args) {
    ProxyCertificate cert;

    System.out.println("Running integration test #2");

    try {
        cert = getUserCertificate();
    } catch (GeneralSecurityException e) {
        System.out.println("Failed to get user certificate");
        return;
    }

    GridConnection conn;

    try {
        conn = Grid.authenticate(cert);
    } catch (SystemException e) {
        System.out.println("Couldn't not find the user certificate");
        return;
    } catch (UserException e) {
        System.out.println("Couldn't not authenticate with "
            + "underlying Grid middleware");
        System.out.println(e.getMessage());
        return;
    }

    GridResourceSpecification resspec =
        new GridResourceSpecification();

    GridJobSpecification spec = new GridJobSpecification();

    spec.setResourceSpecification(resspec);
    spec.setWorkingDirectory("/bin");
    spec.setLocation("echo");
    spec.addArgument("Hello Grid World!");
    spec.setStderr(System.err);
    spec.setStdout(System.out);

    GridJob job = null;

    try {
        job = conn.submitJob(spec, "dheghnom.dnsalias.com", 8443);
    } catch (UserException e) {
        System.out.println("Couldn't not authenticate with "
            + "underlying Grid middleware");
        System.out.println(e.getMessage());
        return;
    } catch (SystemException e) {
        System.out.println("An error occured with out/input "
            + "redirection or the connection to the job "
            + "broker failed.");
        System.out.println(e.getMessage());
        return;
    }

    while (job.getStatus() != GridJobStatus.DONE)
    {
            ;
        }
        System.out.println("TEST PASSED!");
    }
}
```

input-data.txt:

```
Abcdefghijklmnopqrstuvwxyz
```

StringReverser.java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 * A simple string reverser.
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class StringReverser {

    /**
     * A reserve function
     *
     * @param str
     *          The string to reverse
     * @return the reserved string
     */
    private static String reverse(String str) {
        StringBuilder sb = new StringBuilder(str);

        return sb.reverse().toString();
    }

    /**
     * Main function.
     *
     * @param args
     */
    public static void main(String[] args) {
        FileReader reader = null;
        FileWriter writer = null;

        try {
            reader = new FileReader("/home/kenneth/input-data.txt");
        } catch (FileNotFoundException e) {
            System.out.println("Cannot /home/kenneth/find input-data.txt");
        }

        BufferedReader in = new BufferedReader(reader);

        String str = null;
        try {
            str = in.readLine();
        } catch (IOException e) {
            System.out.println("Couldn't read the file");
        }

        String revStr = reverse(str);

        File output = new File("/home/kenneth/output-data.txt");

        try {
            output.createNewFile();

            writer = new FileWriter(output);

            writer.write(revStr, 0, revStr.length());
            writer.flush();
            writer.close();
        } catch (IOException e) {
            System.out.println("Couldn't create "
                + "/home/kenneth/output-data.txt");
        }
    }
}
```

IntegrationTest_3.java:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;
import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.rce.sdk.grid.Grid;
import de.rcenvironment.rce.sdk.grid.GridConnection;
import de.rcenvironment.rce.sdk.grid.GridJob;
import de.rcenvironment.rce.sdk.grid.GridJobStatus;
import de.rcenvironment.rce.sdk.grid.GridJobSpecification;
import de.rcenvironment.rce.sdk.grid.GridResourceSpecification;

/**
 * Integration test 3: Execution of Java-based Grid Job.
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class IntegrationTest_3 {

    /**
     * The service contact.
     */
    private static final String SERVICE_CONTACT = "dheghnom.dnsalias.com";

    /**
     * Period of validity.
     */
```

```
private static final long PERIOD_OF_VALIDITY = 99999990;

/**
 * The path to the users' home directory.
 */
private static final String HOME_PATH =
    System.getProperty("user.home");

/**
 * The system-specific file separator.
 */
private static final String FILE_SEPARATOR =
   System.getProperty("file.separator");

/**
 * Gets user certificate by reading it from the hardarive.
 *
 * @return an RCE ProxyCertificate
 * @throws GeneralSecurityException
 */
public static ProxyCertificate getUserCertificate() throws
    GeneralSecurityException {
    InputStream inputStream = null;
    try {
        inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
            + ".globus" + FILE_SEPARATOR + "usercert.pem");
    } catch (FileNotFoundException e) {
        System.out.println("User Certificate not found: " + HOME_PATH
            + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
            + "usercert.pem");
    }

    X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

    Date timestamp = new Date();
    timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

    return new ProxyCertificate(x509Cert, timestamp);
}

/**
 * Main function.
 *
 * @param args
 */
public static void main(String[] args) {
    ProxyCertificate cert;

    System.out.println("Running integration test #3");

    try {
        cert = getUserCertificate();
    } catch (GeneralSecurityException e) {
        System.out.println("Failed to get user certificate");
        return;
    }

    GridConnection conn;

    try {
        conn = Grid.authenticate(cert);
    } catch (SystemException e) {
            System.out.println("Couldn't not find the user certificate");
            System.out.println(e.getMessage());
            return;
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        }

        GridResourceSpecification resspec =
            new GridResourceSpecification();

        GridJobSpecification spec = new GridJobSpecification();

        spec.setResourceSpecification(resspec);
        spec.setWorkingDirectory("/usr/bin");
        spec.setLocation("java");
        spec.addArgument("StringReverser");
        spec.addEnvironmentVariable("CLASSPATH",
            "$CLASSPATH:/home/kenneth/");
        spec.setStderr(System.err);
        spec.setStdout(System.out);
        spec.addStageInFile("/home/kenneth/workspace/" +
            "RCEIntegrationTests/StringReverser.class",
            "/home/kenneth/StringReverser.class");
        spec.addStageInFile("/home/kenneth/workspace/" +
            "RCEIntegrationTests/input-data.txt",
            "/home/kenneth/input-data.txt");
        spec.addStageOutFile("/home/kenneth/output-data.txt",
            "/home/kenneth/workspace/" +
            "RCEIntegrationTests/output-data.txt");

        GridJob job = null;

        try {
            job = conn.submitJob(spec, SERVICE_CONTACT, 8443);
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        } catch (SystemException e) {
            System.out.println("An error occured with out/input "
                + "redirection or the connection to the job "
                + "broker failed.");
            System.out.println(e.getMessage());
            return;
        }

        while (job.getStatus() != GridJobStatus.DONE)
        {
            ;
        }
        System.out.println("TEST PASSED!");
    }
}
```

Counter.java:

```java
/**
 * A simple counter.
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class Counter {

	/**
	 * Main function.
	 *
	 * @param args
	 * @throws InterruptedException
	 */
	public static void main(String[] args) throws InterruptedException {

		int count = 0;

		while (count < 10) {
			count++;
			Thread.sleep(200);
			System.out.println(count);
		}
	}
}
```

IntegrationTest_4.java:

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.rce.sdk.grid.Grid;
import de.rcenvironment.rce.sdk.grid.GridConnection;
import de.rcenvironment.rce.sdk.grid.GridJob;
import de.rcenvironment.rce.sdk.grid.GridJobStatus;
import de.rcenvironment.rce.sdk.grid.GridJobSpecification;
import de.rcenvironment.rce.sdk.grid.GridResourceSpecification;

/**
 * Integration test 4: Pausing and resuming for
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class IntegrationTest_4 {

	/**
	 * Period of validity.
	 */
```

```
private static final long PERIOD_OF_VALIDITY = 99999990;

/**
 * The path to the users' home directory.
 */
private static final String HOME_PATH =
    System.getProperty("user.home");

/**
 * The system-specific file separator.
 */
private static final String FILE_SEPARATOR =
   System.getProperty("file.separator");

/**
 * Gets user certificate by reading it from the harddrive.
 *
 * @return an RCE ProxyCertificate
 * @throws GeneralSecurityException
 */
public static ProxyCertificate getUserCertificate() throws
    GeneralSecurityException {
    InputStream inputStream = null;
    try {
        inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
            + ".globus" + FILE_SEPARATOR + "usercert.pem");
    } catch (FileNotFoundException e) {
        System.out.println("User Certificate not found: " + HOME_PATH
            + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
            + "usercert.pem");
    }

    X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

    Date timestamp = new Date();
    timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

    return new ProxyCertificate(x509Cert, timestamp);
}

/**
 * Main function.
 *
 * @param args
 */
public static void main(String[] args) {
    ProxyCertificate cert;

    System.out.println("Running integration test #4");

    try {
        cert = getUserCertificate();
    } catch (GeneralSecurityException e) {
        System.out.println("Failed to get user certificate");
        return;
    }

    GridConnection conn;

    try {
        conn = Grid.authenticate(cert);
        } catch (SystemException e) {
            System.out.println("Couldn't not find the user certificate");
            System.out.println(e.getMessage());
            return;
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        }

        GridResourceSpecification resspec = new
           GridResourceSpecification();

        GridJobSpecification spec = new GridJobSpecification();

        spec.setResourceSpecification(resspec);
        spec.setWorkingDirectory("/usr/bin");
        spec.setLocation("java");
        spec.addArgument("Counter");
        spec.addEnvironmentVariable("CLASSPATH",
            "$CLASSPATH:/home/kenneth/");
        spec.setStderr(System.err);
        spec.setStdout(System.out);
        spec.addStageInFile("/home/kenneth/workspace/" +
            "RCEIntegrationTests/Counter.class",
            "/home/kenneth/Counter.class");

        GridJob job = null;

        try {
            job = conn.submitJob(spec, "dheghnom.dnsalias.com", 8443);
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        } catch (SystemException e) {
            System.out.println("An error occured with out/input "
                + "redirection or the connection to the job "
                + "broker failed.");
            System.out.println(e.getMessage());
            return;
        }

        try {
            job.suspend();
            job.resume();
        } catch (SystemException e) {
            e.printStackTrace();
        } catch (UserException e) {
            e.printStackTrace();
        }

        while (job.getStatus() != GridJobStatus.DONE)
        {
            ;
        }
        System.out.println("TEST PASSED!");
    }
}
```

IntegrationTest_5.java:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.rce.sdk.grid.Grid;
import de.rcenvironment.rce.sdk.grid.GridConnection;
import de.rcenvironment.rce.sdk.grid.GridJob;
import de.rcenvironment.rce.sdk.grid.GridJobSpecification;
import de.rcenvironment.rce.sdk.grid.GridJobStatus;
import de.rcenvironment.rce.sdk.grid.GridResourceSpecification;

/**
 * Integration test 6: Enforcing Resource Requirements.
 *
 * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class IntegrationTest_5 {

    /**
     * Period of validity.
     */
    private static final long PERIOD_OF_VALIDITY = 99999990;

    /**
     * The path to the users' home directory.
     */
    private static final String HOME_PATH =
        System.getProperty("user.home");

    /**
     * The system-specific file separator.
     */
    private static final String FILE_SEPARATOR =
       System.getProperty("file.separator");

    /**
     * Gets user certificate by reading it from the harddrive.
     *
     * @return an RCE ProxyCertificate
     * @throws GeneralSecurityException
     */
    public static ProxyCertificate getUserCertificate() throws
        GeneralSecurityException {
        InputStream inputStream = null;
        try {
            inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
                + ".globus" + FILE_SEPARATOR + "usercert.pem");
        } catch (FileNotFoundException e) {
            System.out.println("User Certificate not found: " + HOME_PATH
                + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
                + "usercert.pem");
        }

        X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

        Date timestamp = new Date();
        timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

        return new ProxyCertificate(x509Cert, timestamp);
    }

    /**
     * Main function.
     *
     * @param args
     */
    public static void main(String[] args) {
        ProxyCertificate cert;

        System.out.println("Running integration test #5");

        try {
            cert = getUserCertificate();
        } catch (GeneralSecurityException e) {
            System.out.println("Failed to get user certificate");
            return;
        }

        GridConnection conn;

        try {
            conn = Grid.authenticate(cert);
        } catch (SystemException e) {
            System.out.println("Couldn't not find the user certificate");
            System.out.println(e.getMessage());
            return;
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        }

        GridResourceSpecification resspec = new
            GridResourceSpecification();

        try {
            resspec.setNoOfProcessors(1);
        } catch (UserException e1) {
            e1.printStackTrace();
        }

        GridJobSpecification spec = new GridJobSpecification();

        spec.setResourceSpecification(resspec);
        spec.setWorkingDirectory("/bin");
        spec.setLocation("echo");
        spec.addArgument("Hello Grid World!");
        spec.setStderr(System.err);
        spec.setStdout(System.out);

        boolean passed = conn.queryResources(resspec,
            "dheghnom.dnsalias.com", 8443);
        if (passed) {
            System.out.println("We passed the resource requirements");
        } else {
            System.out.println("We didn't pass the resource requirements");
            return;
        }

        GridJob job = null;

        try {
            job = conn.submitJob(spec, "dheghnom.dnsalias.com", 8443);
        } catch (UserException e) {
            System.out.println("Couldn't not authenticate with "
                + "underlying Grid middleware");
            System.out.println(e.getMessage());
            return;
        } catch (SystemException e) {
            System.out.println("An error occured with out/input "
                + "redirection or the connection to the job "
                + "broker failed.");
            System.out.println(e.getMessage());
            return;
        }

        while (job.getStatus() != GridJobStatus.DONE)
        {
            ;
        }
        System.out.println("TEST PASSED!");
    }
}
```

IntegrationTest_6.java:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.security.GeneralSecurityException;
import java.security.cert.X509Certificate;
import java.util.Date;

import org.globus.gsi.CertUtil;

import de.rcenvironment.rce.sdk.common.ProxyCertificate;
import de.rcenvironment.rce.sdk.exception.SystemException;
import de.rcenvironment.rce.sdk.exception.UserException;
import de.rcenvironment.sdk.grid.Grid;
import de.rcenvironment.sdk.grid.GridConnection;
import de.rcenvironment.sdk.grid.GridDataManagement;

 /**
  * Integration test 6: Playing with files
  *
  * @version $LastChangedRevision$
 * @author Kenneth Rohde Christiansen
 */
public class IntegrationTest_6 {

    /**
     * Period of validity.
     */
    private static final long PERIOD_OF_VALIDITY = 99999990;

    /**
     * The path to the users' home directory.
     */
    private static final String HOME_PATH =
        System.getProperty("user.home");

    /**
     * The system-specific file separator.
     */
    private static final String FILE_SEPARATOR =
       System.getProperty("file.separator");

    /**
     * Gets user certificate by reading it from the harddrive.
     *
     * @return an RCE ProxyCertificate
     * @throws GeneralSecurityException
     */
    public static ProxyCertificate getUserCertificate() throws
        GeneralSecurityException {
        InputStream inputStream = null;
        try {
            inputStream = new FileInputStream(HOME_PATH + FILE_SEPARATOR
                + ".globus" + FILE_SEPARATOR + "usercert.pem");
        } catch (FileNotFoundException e) {
            System.out.println("User Certificate not found: " + HOME_PATH
                + FILE_SEPARATOR + ".globus" + FILE_SEPARATOR
                + "usercert.pem");
        }

        X509Certificate x509Cert = CertUtil.loadCertificate(inputStream);

        Date timestamp = new Date();
        timestamp.setTime(timestamp.getTime() + PERIOD_OF_VALIDITY);

        return new ProxyCertificate(x509Cert, timestamp);
    }

    /**
     * Main function.
     *
     * @param args
     */
    public static void main(String[] args) {
        ProxyCertificate cert;

        System.out.println("Running integration test #6");

        try {
            cert = getUserCertificate();
        } catch (GeneralSecurityException e) {
            System.out.println("Failed to get user certificate");
            return;
    }

    GridConnection conn;

    try {
        conn = Grid.authenticate(cert);
    } catch (SystemException e) {
        System.out.println("Couldn't not find the user certificate");
        System.out.println(e.getMessage());
        return;
    } catch (UserException e) {
        System.out.println("Couldn't not authenticate with "
            + "underlying Grid middleware");
        System.out.println(e.getMessage());
        return;
    }
    GridDataManagement management = new GridDataManagement(conn);

    // create local file
    // write to file

    String fileName = "/home/kenneth/fileToPut.txt";

    System.out.println("Creating with " + fileName
         + " with the content \"hello grid world\"");

    File fileToPut = new File(fileName);
    BufferedWriter bw;

    try {
        fileToPut.createNewFile();
        bw = new BufferedWriter(new FileWriter(fileToPut));

        String[] words =  {"hello", "grid", "world" };

        for (String word: words) {
            bw.write(word + " ");
        }
        bw.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }

    // copy (put) file to grid node
    // list directory

    String gsiport = "5678";
    String gsiserver = "dheghnom.dnsalias.com";

    String address = "gsiftp://" + gsiserver + ":" + gsiport;

    System.out.println("We copy the file to the grid node "
        + "and list the directory");
    String[] files = null;

    try {
        management.put(fileToPut, address
            + "/home/kenneth/grid/greeting.txt", false);

        files = management.listDirectory(address
            + "/home/kenneth/grid/");
    } catch (UserException e) {
            e.printStackTrace();
        } catch (SystemException e) {
            e.printStackTrace();
        }

        System.out.println("The directory contains the "
            + "following file(s):");

        for (String file: files) {
             System.out.print(file + " ");
        }
        System.out.println();


        // create new local file
        // copy (get) file from grid node
        // read / print contents of the file

        fileName = "/home/kenneth/fileToGet.txt";

        System.out.println("Creating the file " + fileName
            + " which will be used to receive contents");

        File fileToGet = new File(fileName);

        try {
            fileToGet.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            management.get(fileToGet, address +
                "/home/kenneth/grid/greeting.txt", false);
        } catch (UserException e) {
            e.printStackTrace();
        } catch (SystemException e) {
            e.printStackTrace();
        }

        BufferedReader br;

        System.out.println("Reading and printing out the content of "
              + fileToGet);
        try {
            br = new BufferedReader(new FileReader(fileToGet));
            String line;

            while (true) {
                if ((line = br.readLine()) == null) {
                    break;
                }
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("TEST PASSED!");
    }
}
```