UNIVERSITY OF AMSTERDAM

Middleware independent workflow engine

Author:
Tünde BÁLINT
(Student number: 5902134)

Supervisor:
Mr. Adam S.Z.BELLOUM - University of Amsterdam
Mr. Oscar KOEROO - Nikhef

October 4, 2010

Abstract

Modern science and engineering are increasingly done in a collaborative fashion. These collaborations are multi-institutional, multi-disciplinary and geographically distributed environments. The Grid is the collection of physical and logical resources such as computing and storage facilities, file systems and high performance networks to which shared access is mediated by grid protocols. A number of middleware implementations have been proposed and implemented that interface the user applications executing on the Grid infrastructure. Thus, the majority of the Grids work in isolation, because the exposed interfaces are incompatible. It is common for the resource needs of grid applications to go beyond what is available in any of the sites making up a Grid. In order for a user to use several grids, running different middleware system, he/she has to be familiar with all the details of these system.

In this thesis we look into the challenging requirements for virtualizing the grid middleware systems and enabling clients to submit to several Grids with one unified interface. The developed application is easily extensible, thus allowing the integration with other middleware systems too. Managing credentials, jobs and data is done in a unified way. At the moment the gMInION abstract layer allows users to access the Grid resources which use the gLite or Globus middleware. An experimental evaluation of access latencies and added overhead is presented. These results show that the overheads introduced for using the new gMInION services are acceptable. The developed system can be a valuable asset for individual clients or for workflow manager systems and further research into Grid middleware independent submission systems.

Acknowledgments

First and foremost I would like to express my sincerest gratitude to Oscar Koeroo, my supervisor at the Dutch National Institute for Subatomic Physics, for his patience, kindness and the guidance he has given me. He has provided me with enthusiastic and encouraging support, always reminding me to look at the big picture, be thorough, but not get lost in the details. He always took the time to answer all my questions patiently and provide me with valuable information, giving me insightful suggestions during the architecture design and helping me with the technical problems I ran into.

I would also like to thank Adam Belloum, my supervisor at the University of Amsterdam for the suggestions during the requirements analysis, thesis writing and presentational skills. His guidance was valuable throughout the whole project.

I am grateful to Sylvain Reynaud for his prompt response to all my questions and valuable help regarding JSAGA. I would also like to thank Dmitry Vasyunin for all his help regarding WS-VLAM and to Reggie Cushing for his collaboration.

Many thanks to all my colleagues at Nikhef for their good company, encouragement and helpful discussions when I encountered weird errors. Their presence has certainly lightened up my life.

Furthermore, I would also like to thank Jutka Hatházi, Anna-Maria Hatházi and Tim Alkemade for their love, support and motivation during the last years or my studies.

Contents

List of Figures

List of Tables

Preface

Motivation

Many scientific experiments require the coordination of a large number of tasks and/or the access and management of large amounts of data in a secure way using a distributed environment. Grid systems can provide the necessary resources for these requirements. When dealing with the interaction and management of complex tasks users are hindered by the gap between different Grid middleware systems and their scientific application. The same obstacles are stumbled upon in case of data storage and management. Due to the complexity of the Grid infrastructures and the evolving standards, abstractions are needed to hide the problems originated from the different Grid middleware architectures.

Workflow frameworks helps scientists to describe their work in a concise way. They also provide abstractions to use the Grid systems, but these abstractions are still limited by the constraint that they usually can access only a specific Grid middleware system. Submitting and managing these workflows independent of the underlying Grid middleware is still a research area.

Problem statement

The goal of this thesis is to study and develop new and generic methods for virtualizing Grid computing resources. An abstract layer has to be designed and implemented which addresses the problems encountered during a job's life cycle in several existing Grid middleware systems (gLite, Globus) and which can provide a user-friendly and unified way to gain access to all the available resources. The developed system should be easily extensible and it should provide transparent access for the users. Workflow submission, as well as problems encountered due to the data dependencies between the workflow tasks need to be solved in a consistent way.

Resource scheduling should be taken into account, allowing the user to see multiple resources he/she can use to submit jobs.

The following requirements should be met:

1. Provide a middleware agnostic abstraction layer for job and workflow submission.
2. Identify which middleware system should be used for a given job, taking into account the available security tokens.
3. Identify resources which can be used to submit jobs, taking into account the available and usable middleware systems and the user's access rights.
4. Submit jobs and retrieve outputs to/from the underlying middleware.
5. Monitor the state of the jobs during the execution

The resulting application should be able to improve the workflow submission in systems such as WS-VLAM, ensuring that these can reliably access the existing Grid resources, independent of the underlying middleware.

In this thesis the main goal is to develop a submission system for workflows, which provides an abstraction, a virtualization of service capabilities. The middleware abstraction layer should be able to handle components for job (workflow) submission, control and file management.

Contribution of the work

The middleware agnostic layer was developed within the WS-VLAM project, which is an e-Science Workflow Management System designed to orchestrate workflows.

The abstract job submission module alleviates one of the shortcomings of this system, namely: the WS-VLAM project could only take advantage of the resources running the Globus middleware. The abstract layer is only responsible for submitting and managing the jobs and the corresponding files. The workflow orchestration, respectively determining the dependencies between the jobs is done by other modules of WS-VLAM.

This thesis was developed in collaboration with Nikhef, the Dutch National Institute for Subatomic Physics. From this perspective it contributes by bringing a broad literature study of the available systems which are able to submit jobs independent of the underlying Grid middleware systems. It is considered useful, because such systems would alleviate the coupling between the client applications and the underlying infrastructure. Using such a system would allow changing the underlying infrastructure, without influencing and involving the clients.

A more practical perspective is that WS-VLAM is being used on the BiG Grid (the National Grid Initiative) resources. To run the jobs submitted with WS-VLAM, the LCG Computing Element [26] (LCG-CE) has to be used.[1] The LCG-CE offers access to the computing resources using the Globus Gatekeeper and GRAM protocols. To improve performance a new computing element was developed, called Computing Resource Execution And Management [5, 6] (CREAM CE) which should replace the LCG-CE. One disadvantage is that the new system requires a different methods to submit jobs. Currently development is done to upgrade all the client to use the new computing element. The LCG-CE is still maintained, but as soon as all clients manage to adapt their applications to use the new, improved CREAM CE the services offered by LCG-CE will be made inaccessible.

Structure of the text

This document is organized as follows.

First an overview is given, in chapter 1, about eScience and its interaction with the Grid technologies. It also presents a general introduction to the field of Grid computing and underlines some of the problems which occur when dealing with this environment. The concept of scientific workflows is also outlined.

Chapter 2 is a literature study presenting and comparing other project which are relevant to our project. The first section (2.1) presents two middleware systems, gLite and Globus. The main components of these systems are described to underline the architectural differences between these middleware systems. The goal of this section is

[1] The computing element is the service representing a computing resource. Its main functionality is job management (job submission, job control, etc.).

to prove that using multiple middleware systems without an abstract layer requires a lot of work. Users familiar with the two middleware systems can skip this section. The discussion goes into section 2.2 which describes other application toolkits that tried to address the job submission problem to several middleware systems. These systems can provide useful insight into the problems which were already addressed. The comparison at the end of the chapter underlines that none of these systems were meant to be used by workflow engines.

Chapter 3 is the most important, since it describes the design and implementation decisions made regarding the architecture of the new abstract layer. It elaborated the functional and non-functional requirements (Section 3.1), discusses the used design patterns (Section 3.2.2) and presents the proposed architecture (Section 3.3). It also describes some security issues which need to be addressed (Section 3.2.3) and details the experiences and problems encountered during the implementation phase.

Chapter 4 presents the performance testing setup and analyses the obtained results. It also outlines the integration with the WS-VLAM workflow management system (Section 4.6).

In chapter 5 we present a summary of the work and suggestions are presented for future improvements.

Chapter 1

Introduction

The complexity of science, in order to achieve the best results, often requires the worldwide collaboration of multidisciplinary teams. e-Science enhances science by promoting collaborative research using computers and communication technology. It employs computationally-intensive, data-intensive, and highly distributed methods to enable more efficient resource usage and knowledge sharing. e-Science integrates Grid computing, programming tools, data and visualization technologies. This offers scientist tools to generate, analyze, share and discuss their insights, experiments and results.

The Virtual Laboratory concept, as shown in figure 1.1, brings all of the components together in one environment, which a scientist can use. The goal of the Virtual Laboratory for eScience project is stated as follows: "to bridge the gap between the technology push of the high performance networking and the Grid and the application pull of a wide range of scientific experimental applications." [44] Hence the scientists don't interact directly with the middleware, but rather with the Virtual Laboratory which interfaces the middleware.

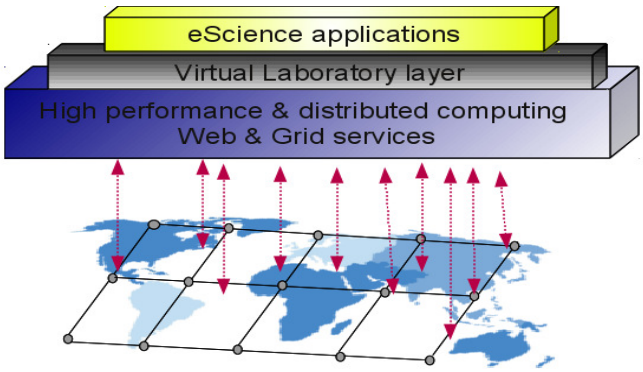


Figure 1.1: Layered eScience architecture, which decouples the scientists' application from the underlying infrastructure. In this way resources are available in a transparent way.

The Virtual Laboratory for eSciene concept makes the life of scientists easier, allowing them to concentrate on their research, instead of dealing with the underlying resources.

The underlying computer infrastructure consists of highly distributed networking environments using high-performance computing platforms, including clusters, Grids and clouds. Since the main focus of this thesis is studying and developing new and generic

methods for virtualizing Grid computing resources, only this environment is elaborated further. The base layer of a Grid system is made up of the available resources, such as supercomputers, clusters of computers, storage networks, specialized database services or valuable scientific instruments. Grid technologies aim at decoupling doing science from location and time, by making resources available in a transparent way using a Grid middleware. This middleware interfaces the resources and local control facilities. It allows scientists to use information or data without consciously being aware that this data is stored far away or maybe in multiple location. Also a large amount of computational resources are accessible, which can provide timely feedback about the jobs which run on them.

Unfortunately, since the formulation of the Grid vision [33], hundreds of Grids have been built in different countries for different sciences. These can serve different proof-of-concept purposes for computer science research or production work. However, the vast majority of these Grids work in isolation, running counter to the fundamental idea of Grids. Running jobs on a given Grid middleware involves addressing problems like: authorization, authentication, file pre- and post-staging, monitoring and data management. Most of the existing Grid systems address all these issues, but they provide different ways to obtain the same results. When dealing with the interaction and management of complex tasks users are hindered by the gap between different Grid middleware systems and their scientific application. The same obstacles are stumbled upon in case of data storage and management.

To meet more complex goals, and to alleviate the process of creating complex experiments, workflows can be used to combine and coordinate services. These workflows aim at the automation of scientific processes based on data dependencies and their control. They constitute a suitable format to exchange applications among developers, end-users and Grid experts. Not only do they provide a intuitive representation of the logic of the application to the end-user, but also allow parallelization of the code, since workflow components can be run at the same time, as long as they are not interdependent. Workflow management systems are becoming very popular among scientists who want to use complex computing infrastructure to perform their enhanced science (eScience). [42, 58, 10, 24] These workflow systems are part of the virtual laboratory layer in figure 1.1. However, existing workflow management systems do not completely fit with the current vision of the Grid technology and thus need to be redesigned. In most cases a workflow management system only supports a limited set of Grid middleware systems, thus narrowing the number of accessible resources. [25]

1.1 Grid Computing

Grid technologies [33, 31] are becoming more and more applicable. The main aim of Grid computing is to enable secure sharing of resources between organizations and therefore across administrative domains. Using Grid technologies, several computers are used to solve the same scientific or technical problem, which requires a great number of computer processing cycles or access to large amounts of data. Most users of these systems wish to maintain the privacy of their documents, although they are accessing resources across multiple administrative domains, whilst they also require easy access to these resources.

Grids are large scale shared distributed systems. Distributed computational systems consist of diverse end systems. These end systems typically consist of computer systems like CPU and memory, storage elements and they are connected via high speed networks. To use the available Grid resources, software is used to divide and apportion pieces of a program among several computers, sometimes up to many thousands. Grid computing

can also be thought of as distributed and large-scale cluster computing, as well as a form of network-distributed parallel processing.

What distinguishes Grid computing from conventional cluster computing systems is that Grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Also, while a computing Grid may be dedicated to a specialized application, it is often constructed with the aid of general-purpose Grid software libraries and middleware.

The dream of a universal Grid, based on a single infrastructure with only one middleware deployed, seems to be unlikely to be turned into reality due to both technical and political issues. However, existing infrastructures having different middleware systems, need to communicate (to be "interoperable") between each other in order to provide more resources to users.

Many middleware systems have been developed to support the Grid paradigm. Although these middleware systems aim to implement almost the same functionalities, they have many differences in relation to key aspects such as security, application integration, communication mechanism, etc. However, only some middleware systems have the code maturity to be deployed on real Grid infrastructure. These include, Condor [59] , gLite [15], Globus [30], UNICORE [41] and some others.

In the last years a big effort was put into allowing different middleware systems to interoperate. The Open Grid Forum (OGF) [3] has developed several standards that, if adopted in every middleware, should allow a complete interoperability. However, the adoption of these standard is quite limited, because they have a big impact on the architecture and features of the middleware, so currently the interoperation is indeed very small.

Currently, when a Grid infrastructure is deployed, a given middleware is chosen and then this infrastructure can be used only by clients supporting that specific middleware. Users who want to access the infrastructure need to learn how the clients work, and this can require a lot of time. If users knowledge background is about a different middleware, they need to learn everything again wasting a huge amount of time.

Because these systems aren't fully standardized yet, implementing applications which can be run independently of the middleware is a challenge.

There is a gap between the availability of the Grid and the Grid applications which can use the existing middleware. This is mostly due to the fact that Grid computing is still a research area, so middleware implementations tend to focus on technical issues, and provide only command line interfaces or platform specific interfaces. But for users these technical details are less relevant and exposing them is counter productive. The users only want to run their jobs, workflows and obtain the desired result quickly (even if the jobs require a lot of computing power, or data access). These researchers want to deal with the problems in their own domain of research, not with the technical details of Grid computing. Accessing Grid middleware systems is cumbersome, non user-friendly and mostly requires knowledge of the commands specific to each middleware. In addition to this drawback, the interfaces which are exposed change quite frequently, offering their functionalities in different ways. Thus the users are exposed to the risks of needing to adapt their existing applications to the needs of the interfaces. It is also important to take into account the portability and interoperability problems when we want to access different type of middleware. The incompatibilities between failures and configurations require applications to implement the same solution in different ways just to overcome the heterogeneity of the available middleware.

The different middleware systems support different job submission mechanism. In order for a user to take advantage of all these system, he/she has to be familiar with the raw job submission commands. He/She also has to take into account that some of the used services aren't available on all sites, or that maybe some of them won't work for all

of the users, because of security constraints or site-specific policies. For example some specific certificate attributes can be required to run a given job on some sites. There are also policies which specify which resources can be used by a job, how the job should be scheduled compared to other jobs.

As stated the goal of this thesis is to provide an abstraction, to virtualize Grid computing resources, enabling workflow submission systems to access these resources in an intuitive way.

1.2 Grid Workflows

Building and managing complex scientific experiments, which need to process large data sets or involve lots of computational resources, often require the ability to compose and execute workflows. [25] A workflow [68] is used to pass files and data between participating components according to a set of rules, in an automatic way, to achieve a predefined goal. So basically it is a way of connecting multiple tasks according to their dependencies. It is an automation of the processes, which involve the orchestration of a set of Grid services and agents which must be combined together to solve a problem or define a new service.

A workflow management system is responsible of defining, managing and executing the workflows on computing resources. There are several advantages of using Grid computing to run these workflows. For example:

- it is possible to use specific resources from different administrative domains
- it is possible to use resources located in a particular domain to increase throughput
- it is possible to build dynamic applications which orchestrate distributed resources.

A Grid workflow system based on the workflow reference model [40] should have build-time functions, which deal with defining the workflow tasks and their dependencies and run-time functions which manage the workflow execution and interaction with Grid resources. Information retrieval is also crucial, since a Grid workflow management system only chooses suitable resources to execute the tasks. This means that it should know some static information (e.g. number of processors) and/or dynamic information (e.g. available disk space) and/or historic information (obtained from previous events) about the resources.

Another important problem is caused by the fact that a Grid workflow management system has to be able to coordinate several Grid site specific local management systems, since workflow scheduling should be done in such a way that the underlying middleware isn't taken into consideration. This problem occurs, because Grid resources are heterogeneous in terms of local configuration and policies. The scheduling of a workflow is even more complicated, since there are several possibilities how the scheduling of tasks can be done. One can make scheduling decisions taking into account only the tasks, sub-workflows or taking into account the whole workflow.

Fault tolerance has to be taken into account too. A workflow execution failure can occur for various reasons. Some of these reasons can be: variations in the execution environment, environment configuration, non-availability of required services, overloaded resource conditions or system running out of memory. A Grid workflow management system should be able to identify and handle these cases and support reliable execution in the presence of concurrency and failures. This can be achieved by retrying, checkpointing the workflow, replicating (redundancy) or changing the resource.

It is also important to transfer the pre- and post-staged files. This can be done in multiple ways. The first way consists of transferring the data via a central point. Using the second alternative, the locations of the intermediate data is managed by a distributed data management system. The last alternative consists of the peer-to-peer approach, in which the data transfers occur between resources. This approach encounters additional drawbacks in a Grid environment, since in this case every Grid node has to able to send data to another node.

One has to pay attention also to the problems which are caused by security issues. It has to be taken into account that not all resources are protected adequately. This means that trusting the resources needed to run a complex workflow could result in our delegated identity being compromised.

Although workflows are an efficient way to separate the business logic from the invocation of the appropriate IT resources, we can see that there are still many challenges which need to be address especially if the workflow management systems are dealing with multiple Grid middleware environments.

Chapter 2

Related work

The purpose of this section is to show the main design differences between Grid middleware systems, taking into account security, job submission and data management. A large variety of Grid middleware projects have been conducted in the past years for different purposes. For a researcher not familiar with a Grid environment it is difficult to choose a particular middleware among the available solutions. In this chapter we try to provide a short insight to two of the most commonly used Grid middleware systems: gLite and Globus. We also provide a comparison between the available toolkits which try to address the problems encountered during middleware independent job submission.

2.1 Differences between two middleware systems Globus Toolkit and gLite

This section tries to underline the differences between the Globus Toolkit [30, 29] (GT) and gLite [15] middleware systems. First the two systems are shortly presented. After that we summarize how these systems address issues related to security, data management, information retrieval and job submission. This comparison is useful, because the discussed issues also need to be addressed by the new abstraction layer for each used middleware. It also gives an indication of the amount of knowledge a user has to posses if he/she wants to use a Grid middleware system.

The GT consists of a collection of services which can be deployed to provide a working Grid. These consist of information services, data management services and computing resource services. From version 3 of the GT, web services are used, which operate over HTTP and HTTPS on the standard ports 80 and 443. Web services bring certain benefits such as implementation independent and machine readable protocol specifications in the form of Web Service Description Language [19] (WSDL) and usually firewall restrictions won't interfere with it. By default, standard web services are stateless, which sometimes is considered as a restriction for building Grid Services. Therefore, Web Services Resource Framework [20, 21] (WSRF) introduces state to a web service by extending standard web service technologies such as WSDL and Simple Object Access Protocol [38] (SOAP). It additionally defines the conventions for managing the life cycle of services, the discovery, inspection, and interaction with stateful resources in standard and interoperable ways.

gLite is a middleware that enables resource sharing on the Grid by providing a set of integrated services to allow secure job execution, information retrieval and data management in a distributed Grid environment. The access point to the gLite middleware is the so called User Interface that provides command line tools and APIs for secure job

submission, data management as well as access to the information system. Similar to the Globus Toolkit, information and monitoring services publish and consume information concerning resources, data management services deal with file transfers, while job management services address job execution and workload management.

Security

Security has to be taken into account to allow authorization and authentication of the user and of the resources. This way a match can be made between the user jobs and the available resources. Access control has to be taken into account too. [32] Most users of a Grid system don't want their computation and/or data to be compromised by another user. Using the security policies, resources can be set up in such a way that the access to them is restricted. In addition, computing resources can be configured to give priority to jobs belonging to certain groups of users.

The goal of the Grid Security Infrastructure (GSI) is to provide a "single sign-on" capability for users. This means that a user must sign on to the Grid only once per session no matter which resources are used. Another requirement was to allow delegation of authorization from a user to a Grid service acting in the user's behalf. This takes the form of delegation of the user's identity.

GSI contains components which implement different credential formats and protocols and addresses message protection, authentication, delegation and authorization. It supports WS-Security-compliant [13] message-level security with X.509 credentials, username/ password authentication and transport-level security with X.509 credentials. Each user and resource is assumed to have a X.509 public key certificate. Using these a secure channel can be established to protect messages and to support delegation in a secure way.

In GSI, single sign-on and delegation are accomplished by the use of proxy credentials. To implement single sing-on, a new key-pair is generated on the user's system and a certificate is created with the user's private key. The private key of the proxy credential[1] is stored unencrypted and so the user does not need to re-enter the passphrase of their long-lived private key.

Proxy certificate can successfully resume the role of a user certificate if and only if it can be determined that the proxy certificate originated from the user certificate. Verification of the digital signature performed by the proxy certificate can be done using the proxy certificate which can later be traced back to the user certificate. Typically the validity period of the proxy certificate is short in order to limit exposure time in the event that the proxy private key is compromised.

Globus uses X.509 certificates [61] for authentication and supports a variety of authorization systems. The simplest authorization system uses a *grid mapfile* to map certificate subject names to local account names. In this identity-based authorization method, a string representation of the certificate subject name is used to perform this mapping. The string representation can cause problems with names that use unusual distinguished name components and the support for non-ASCII characters is rather poor. The certificate's issuer name is not included in the mapping; it is assumed that different Certificate Authority (CA)s will not issue certificates with the same subject distinguished name to different entities and this uniqueness requirement is enforced by the agreements on CA namespaces. Because of scalability issues attribute-based authorization had to be adopted. One possibility is to use the Community Authorization Service (CAS) [51], which uses

[1] Proxy certificates are certificates signed by the user, or by another proxy, and they are intended for short-term use, when the user is submitting many jobs and it would be inconvenient for him to repeat his password for every job.

the Security Assertion Markup Language (SAML) [50]. SAML defines formats for a number of types of security assertions and a protocol for retrieving those assertions. Other Globus supported alternatives for achieving authorization using the SAML specification are: the Open Grid Service Architecture Authorization Services (OGSA-AuthZ) [67] and GridShib [65]. Virtual Organization Membership Service [35] (VOMS) authorization is also supported, which takes into account the X.509 attribute certificates.

In the context of Globus each virtual organization is expected to set up its own Grid infrastructure. The Globus software is not explicitly aware of Virtual Organization (VO)s since it is assumed that all communicating users and services are within the same VO.

Contrary to Globus, in gLite a user has to be a member of a VO and his/her Grid certificate has to contain the corresponding VOMS attributes for him/her to be able to use the gLite resources. Using a valid Grid certificate and contacting the VOMS server a proxy certificate can be obtained. gLite uses GSI (i.e. X.509 certificates with VO group and role associations of the user) for basic user authentication. The authorization of a user on a specific Grid resource can be done in two different ways:

1. Using the grid-mapfile mechanism, which maps user credentials to local accounts. When a user's request for a service reaches a host, the certificate subject of the user (contained in the proxy) is checked against what is in the local grid-mapfile to find out to which local account (if any) the user certificate is mapped, and this account is then used to perform the requested operation. This method is obsolete, because of scalability issues and it it hard to provide proper mutiple VO affiliations for a user.

2. Using the VOMS and the Local Center Authorization Service (LCAS)/ Local Credential Mapping Service (LCMAPS) mechanism. The VOMS is a system that allows for complementing a proxy certificate with extensions containing information about the VO, the VO groups the user belongs to, and the role the user has. The LCAS service handles authorization requests for a service (such as the computing element), and the LCMAPS provides all local credentials needed for jobs allowed to access and use the resource. The authorization decision of the LCAS is based upon a user's certificate, proxy or extended proxy. The certificate is passed to authorization modules, which grant or deny the access to the resource.

Information services

The information services constitute an important component in a Grid middleware system. [23] We present a few scenarios in which these services are indispensable. Every time a job is scheduled, a computing element has to be chosen. The specification of the computing element have to match the job description requirements. Interrogating the information service static (available CPUs, memory) and dynamic (waiting time, response time) information can be obtained about regarding the computing element. Another example is when a file has to be replicated. In this case the information system has to contain information about the system configuration and network performance of the storage elements.

It's important to understand how the information services work, because our system should be able to choose the resources to which it can submit to.

In Globus the information services keep track of Grid meta-data for the purpose of monitoring and allowing discovery of data and computation resources. The Monitoring and Discovery Service (MDS) provides support for publishing and querying of resource information. Within MDS, a schema defines the classes that represent various properties of the system. MDS has a three-tier structure at the bottom of which are Information

Providers (IPs) that gather data about resource properties and status and translate them into the format defined by the object classes. The Grid Resource Information Service (GRIS) forms the second tier and is a daemon that runs on a single resource. GRIS responds to queries about the resource properties and updates its cache at intervals defined by the time-to-live by querying the relevant IPs. At the topmost level, the GIIS (Grid Information Index Service) indexes the resource information provided by other GRISs and GIISs that are registered with it. The GRIS and GIIS run on Lightweight Directory Access Protocol (LDAP) backend, which is an open standard. LDAP is a lightweight protocol for accessing directory services optimized for reading, browsing and searching. The LDAP information model is based on entries, which are collections of attributes that have globally-unique distinguished names. Each entry's attributes have a type and one or more values, and entries are arranged in a hierarchical tree-like structure. The standard set of IPs provide data on CPU type, system architecture, number of processors and available memory among others. This information is collected both form computing and storage resources.

Information retrieval in gLite is organized in a hierarchical manner. Figure 2.1 illustrates the hierarchy. At the lowest level, resource-level Berkeley Database Information Index (BDII)s collect information on the state of resources (using scripts called information providers"). Site-level BDIIs aggregate that information, and make it available to the top-level BDIIs. Periodically, the higher level servers make LDAP queries to the lower-level ones. There are multiple instances of the top level BDII in order to provide fault tolerance.

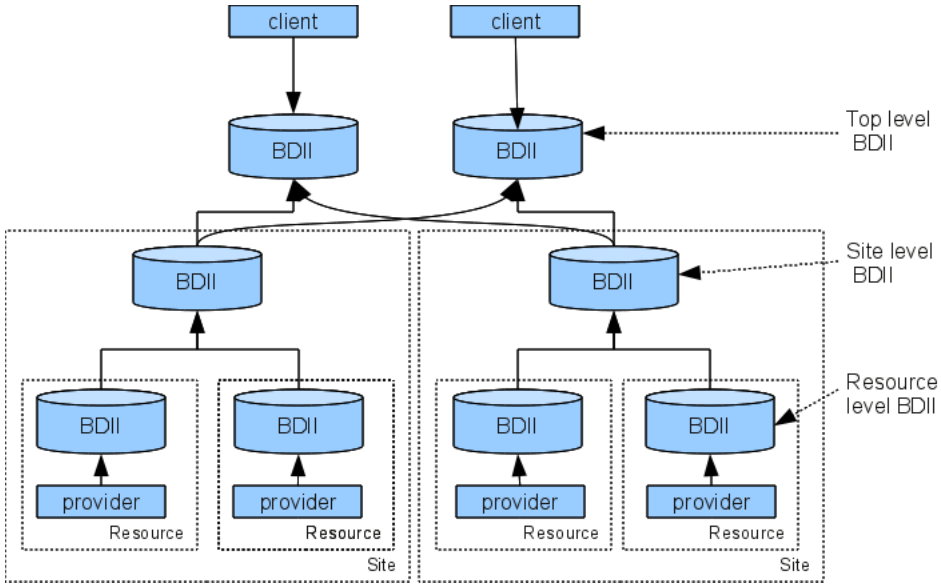


Figure 2.1: gLite Information system hierarchy. Information from the resources is collected periodically. Application can query the information service to find available resources. Data is represented as a hierarchy of objects forming a tree structure.

Similar to the Globus information service, the protocol used to query the gLite information system is LDAP.

To query the gLite BDII or the Globus information services, one needs to understand the layout of the information therein, which is specified by the Grid Laboratory Uniform

Environment [11] (GLUE) schema. The GLUE schema is a data model to describe, in a precise and systematic way, information on static and dynamic Grid resources (including state and VO-specific views).

Data management

Data management services are used to locate and transport data in a secure way between storage systems and applications. Although our work focuses on virtualizing Grid computing resources, we have to provide some file management services too. It is obvious that these services should also be middleware agnostic. These services are useful, for example, when the input and output files are transferred.

In Globus the data components include GridFTP, an enhanced file transfer system supporting third-party transfer, striping[2], and additional security features [7]. The Globus implementation of the GridFTP specification uses GSI and provides libraries for reliable and secure data movement. The Reliable File Transfer service is used to reliably manage multiple GridFTP transfers, meaning that it can orchestrate the transfer of files in an efficient way. The Replica Location Service is a decentralized system which maintains information about the location of replicated files. An application needing to access one instance of a file needs to transparently access such a file, independently of its location and storage properties. The Replica Location Service service provides a mechanism for storing/associating a logical file name to a Grid Universal Identifier (GUID), and to associate a set of physical file handles at the various Grid locations to a GUID. To copy a file using GridFTP the following command is used: globus-url-copy.

Storage elements provide the virtualization of a storage resource in gLite. They consists of the storage back-end with all the associated hardware and drivers, the transfer service for a (set of) transfer protocol(s), gLite File I/O service and auxiliary security and logging services. Each storage element has to implement the Storage Resource Manager (SRM) interface, thus providing a uniform access method to all the storage elements. The catalog services keep track of the data location as well as relevant meta data (e.g. checksums and file sizes) and the data movement services allow for efficient managed data transfers between storage elements. The access to files is controlled by access control lists.

The user application does not need to know where the files are exactly stored, as long as it is able to run and read the data as if it were local. Therefore the name by which the user refers to the file has to be a location independent Logical File Name (LFN)[3]. Upon creation, each file also acquires an immutable GUID[4]. There can be several instances of a given file, which are being tracked by the Replica Catalog. The replicas are identified by Site URL (SURL)s [5]. Each replica has its own SURL, specifying implicitly which storage element needs to be contacted to extract the data. Usually, users are not exposed directly to SURLs, but only to the logical namespace defined by LFNs. The

[2]Data striping is the technique of segmenting logically sequential data, such as a file, in a way that accesses of sequential segments are made to different physical storage devices. Striping is useful when a processing device requests access to data more quickly than a storage devices can provide access. By performing segment accesses on multiple devices, multiple segments can be accessed concurrently.

[3] A logical (human readable) identifier for a file. LFNs are unique but mutable, i.e. they can be changed by the user. The namespace of the LFNs is a global hierarchical namespace.Each VO has its own namespace.

[4] Each LFN also has a GUID (1:1 relationship). GUIDs are immutable, i.e. they cannot be changed by the user. Once a file acquires GUID it must not be changed otherwise consistency cannot be assured. GUIDs are being used by Grid applications as immutable pointers between files. If these should change, the application may suddenly point to a wrong file.

[5]The SURL specifies a physical instance of a file replica. Also referred to as the Physical File Name (PFN). SURLs are accepted by SRM interface of the storage element. A file may have many replicas, hence the mapping between GUIDs and SURLs is one-to-many.

Grid Catalogs provides the mappings needed for the services to actually locate the files. To the user the illusion of a single file system is given.

Job submission

The services described in this section are responsible for distributing and managing the tasks across the Grid resources. Because of the architectural and design differences between Globus and gLite, the interaction with the user is also different. Because our goal is to virtualize Grid resources, addressing job submission is very important. Our application deals with setting up the job description, managing and monitoring the job and obtaining the results in a middleware agnostic way.

The main computing resource component in Globus is the Grid Resource Allocation and Management (GRAM). This uses an HTTP-based protocol. Based on the GRAM protocol, the GRAM service is composed of three sub-parts: a gatekeeper, a job manager and a reporter. The GRAM Gatekeeper receives requests written in a Replica Location Service. To execute the request first mutual authentication is done between the user and the resource, using the GSI service. The request is then forwarded to a job manager, which passes the work to a batch system's queue. The GRAM service provides access to a computer which fulfills the specified job requirements, stages the executable and the specified files, starts the execution of the program and monitors and manages the resulting computation. The job manager terminates as soon as the jobs, for which it is responsible for, have terminated. The GRAM reporter is responsible for storing information about the status and the characteristics of resources and jobs in MDS. The job details are specified through the Globus Resource Specification Language [22, 1] (RSL), which is a part of GRAM. RSL provides syntax consisting of attribute-value pairs for describing resources required for a job including the minimum memory and the number of CPUs. To submit and manage jobs the following commands are useful: globusrun, globus-job-run, globus-job-submit, globus-job-status and globus-job-get-output.

In gLite the Workload Management System (WMS) is a Grid level metascheduler that schedules jobs on the available computing elements according to user preferences and several policies. It also keeps track of the jobs it manages in a consistent way via the logging and bookkeeping service. Jobs to be submitted are described using the Job Description Language [12] (JDL). This can specify which executable to run and its parameters, files to be moved to and from the worker node, the needed input files, and any requirements on the computing element and the worker node. The matchmaking process selects among all available computing elements those which fulfilling the requirements expressed by the user and which are close to specified input files. It then chooses the computing element with the highest rank, a quantity derived from the status information of the computing element. Then the job together with the InputSandbox is transferred to the Gatekeeper of the computing element . The Gatekeeper submits the job to the Local Resource Management System. The Local Resource Management System will then send the job to one of the free worker node of the computing element . When the job has finished, the output files are available on the Local Resource Management System. The job manager running on the computing element notifies the WMS that the job has completed. The WMS subsequently retrieves those files specified in the OutputSandBox. The WMS sends the results (the OutputSandBox) back to the user.

The Logging and Bookkeeping service tracks jobs managed by the WMS. It collects events from many WMS components and records the status and history of the job.

Using the command line interface

In table 2.1 we can see the different commands a user has to be familiar with, if he/she wants to use different middleware systems. These commands achieve (almost) the same end results, but they have different arguments. What's worse, the job description file has to be formatted in a different way and different key-value pairs have to be used. All these systems try to provide the same end result, but sometimes their behavior is slightly different. The error messages are different and the cause of these messages is often hard to understand. In short, getting familiarized with the command line interface of each middleware system is a lot of work and adds burden to the user.

Aspect	Globus	gLite
Security	X.509 certificates and proxy certificates grid-proxy-init grid-proxy-info grid-proxy-destroy	X.509 certificates proxy certificates VOMS voms-proxy-init voms-proxy-info voms-proxy-destroy
Information system	MDS grid-info-search LDAP queries	BDII lcg-infosites, lcg-info LDAP queries
Data management	GridFTP Reliable File Transfer globus-url-copy	LCG File Catalog lcg-cr, lcg-lr, lcg-lg, lcg-rep lcg-aa, lcg-ra, lcg-rf, lcg-uf lg-cp, lcg-la, lcg-del Interact with the SRM srmcp, srmmkdir, srmls srmmv, srmrm, srmrmdir
Job management	GRAM globusrun, globus-job-run, globus-job-submit globus-job-status globus-job-get-output	WMS glite-wms-job-delegate-proxy glite-wms-job-cancel glite-wms-job-status glite-wms-job-output glite-wms-job-info Submit to LCG-CE globus-job-submit globus-job-status globus-job-get-output Submit to CREAM CE glite-ce-delegate-proxy glite-ce-job-submit glite-ce-job-status glite-ce-job-cancel glite-ce-job-list

Table 2.1: Main differences between Globus and gLite commands from a user's perspective

2.2 Application toolkits for job submission

Because we don't want to completely reinvent the wheel, it was considered useful to evaluate the existing application toolkit implementations and decide which one could

contribute to our work. We planned on reusing part of the best implementation.

Multiple approaches have been introduced in attempt to address the middleware independent job submission problem. There is also an API specification which provides a standardized, common interface across various Grid middleware systems and their versions. Unfortunately the implementations of this specification are not mature enough.

Simple API for Grid Applications [37, 36] (SAGA) is a high-level middleware independent API for Grid middleware. It is being standardized by the Open Grid Forum [3] and it uses an object-oriented approach to provide a uniform interface to heterogeneous Grid middleware for security, data management and execution management. It is based on the Distributed Resource Management Application API [52, 60] (DRMAA) , the Grid Application Toolkit [8, 9] (GAT) and the Commodity Grid [64] kit abstraction models.

The job submission part of the SAGA API is based on the DRMAA API. DRMAA is a high-level Open Grid Forum [3] API specification for the submission and control of jobs to one or more distributed resource management systems within a Grid architecture. The scope of the API covers all the high level functionality required for Grid applications to submit, control (terminate, suspend,...), monitor and retrieve jobs to local Grid Distributed Resource Manager systems. It specifies that DRMAA implementations should be provided as shared modules, which may address different distributed resource managers. Late binding is proposed and this way the modules can be interchangeably selected at the run time by the end user by specifying the distributed resource management systems or by setting the environment in the correct way. A difficult problem to address in case of accessing different distributed resource manager systems is the job state semantics, because some states are not implemented in some distributed resource manager systems (e.g. the vanilla universe of the Condor system doesn't support suspension and resuming of jobs). DRMAA doesn't take into account the security aspects of the distributed resource manager systems, but it relies on the security context provided by the user running the application. File staging isn't explicitly addressed in the specification, although when the drmaa_transfer_files job template attribute is supported, then input, output and/or error files are fetched by the distributed resource manager system from the specified host. Resource monitoring and session management aren't addressed. DRMAA only deals with resource management, while the GAT also deals with other important issues, such as Grid I/O, monitoring and information services.

GAT is a language independent object-oriented specification and its goal is to provide a complete application oriented abstraction layer to the underlying Grid middleware. GAT has a modular plugin architecture and it sits between the Grid applications and the Grid middleware systems. JavaGAT [62] is a Java implementation of the GAT API. In JavaGAT the Grid functionality is exposed through GAT objects, by calling methods of the GAT API.

JavaGAT integrates multiple Grid middleware systems with different and incomplete functionality into a single, consistent system, using a technique called intelligent dispatching. [62] This technique dynamically forwards (dispatches) application calls on the JavaGAT API to one or more Grid middleware systems that implement the requested functionality. It has a plug-in architecture, where plug-ins are called adaptors. The selection process of the used adaptor is done at runtime, using late binding. Policies and heuristics are used to automatically select the best available middleware. Using this technique a suitable middleware is selected only when an operation is invoked and not when the corresponding API object is created. This way a single API object can use multiple middleware systems. If a Grid operation fails, the intelligent dispatching feature will automatically select and dispatch the API call to an alternative Grid middleware. This process continues until a Grid middleware successfully performs the requested operation, achieving transparent fault tolerance, or until all the available adaptors were tried.

The most important difference between GAT and JavaGAT is that JavaGAT uses intelligent dispatching, while the other GAT implementations use static dispatching, meaning that the function calls and references are determined at compile time.

In contrast to JavaGAT, DRMAA uses static binding to a particular middleware.

The Commodity Grid is basically the integration of Grid and commodity technologies. In this case Grid mostly means Globus and what is included in Globus. The Java Commodity Grid Kit [64] framework provides abstractions for job executions, file transfers, workflow abstractions, and job queues. Different Grid middleware can be integrated into the Java Commodity Grid kit.

Java Commodity Grid supports dynamic class loading with late binding, so the actual implementation of an API object is selected at run time, but the application must explicitly specify which middleware is used, while with JavaGAT this selection is done automatically. With Java Commodity Grid, a single adaptor, called provider is selected for an entire object, while with JavaGAT intelligent dispatching allows more flexibility and provides the ability to use different middleware to implement a single object. The GAT engine dynamically "routes" the API calls to the respective Grid middleware.

The SAGA specification aims to provide the correct level of abstraction for portable applications across different middleware, meaning that even if different Grid middleware systems are exposing their functionality in different ways, the API assures that these functionalities can be accessed without directly programming against the interfaces exposed. There are several SAGA implementations and they all use the intelligent dispatching technique. Figure 2.2 shows how the SAGA API tries to bridge the gap between infrastructure and applications.

It has to be mentioned that the implementations of the SAGA specification are still under development. No information was discovered about using the SAGA specification for running workflows. All the implementations were designed to be directly used by the client, not by a workflow engine.

As it can be seen in figure 2.2 the applications use the SAGA API and the SAGA engine is responsible of calling the correct adaptors, which would access the underlying middleware systems. This way the infrastructure is virtualized and the user doesn't have to be aware of where his/her jobs are running exactly. In the following paragraphs the C++ and the Java implementations of the SAGA API are presented.

The SAGA C++ [43] implementation is a complete SAGA compliant implementation. It uses an adaptor based late binding architecture trying to provide runtime portability between different backends. A disadvantage is that the number of available adaptors is limited and for example we want to build the Globus Toolkit adaptor, a local installation of the Globus C header and client library files is. The good thing is that the local Globus services don't have to be configured or running. More about the SAGA C++ adaptor can be read in appendix I.

In the following paragraphs the two Java SAGA implementations are going to be presented in more detail.

JavaSAGA [16] is developed at the Vrije University in Amsterdam, Netherlands. It consists of an engine and dynamically loadable adaptors at runtime (late binding). It has only a few adaptors. One of these is JavaGAT, which at its turn can connect to different middleware. Another adaptor which was developed is for GridSAM, which it is possible to submit jobs to resource management systems, such as Condor, PBS and others. Unfortunately the job submission to WMS didn't work, due to some incompatibilities between the JavaSAGA adator for JavaGAT and JavaGAT. More about JavaSAGA can be found in appendix H.

JSAGA [53] is funded by the French National Research Agency(ANR). It is adaptor based. Adaptor interfaces are designed to ease plug-in development and to enable efficient

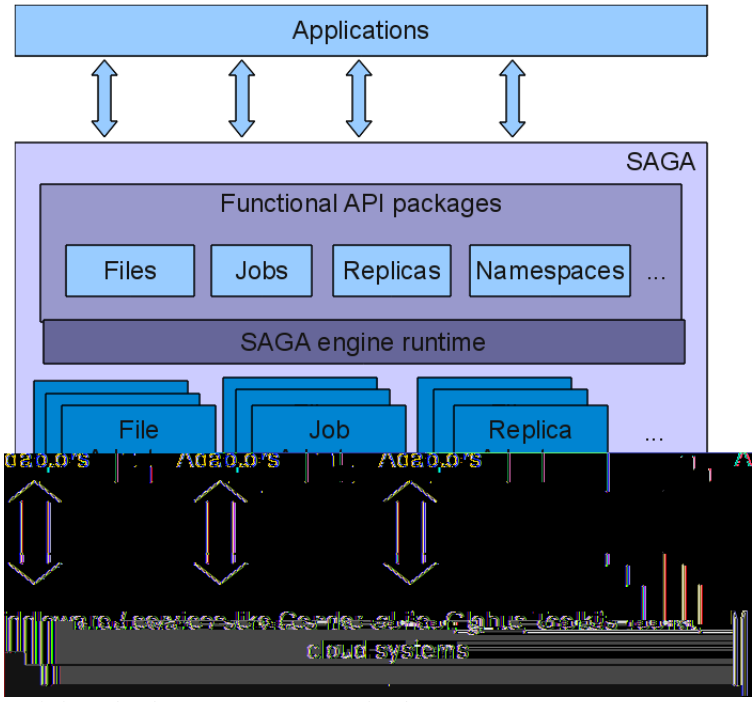


Figure 2.2: SAGA overview. How SAGA tries to bridge the gap between the applications and infrastructure. The SAGA engine is responsible of choosing the suitable middleware and accessing the underlying resources. The user shouldn't be aware of the details of accessed middleware systems.

usage of underlying APIs. These interfaces are service-oriented. JSAGA uses early binding to middleware. The core engine and most adaptors are independent of the operating system, and they do not require any additional package to be installed. For example, one does not need to run JSAGA on a gLite User Interface machine in order to be able to use gLite middleware.

If several security context candidates are available for a given URL, JSAGA throws an exception instead of automatically trying to connect with each of them to find the right one to use. This is done to avoid problems like creating files with unexpected owner, submitting jobs that will be allowed to run but not to store their result, locking accounts because of too many failed connection attempts. More about JSAGA can be found in appendix G.

Further an in-depth comparison is made between JSAGA and JavaSAGA. These two APIs were chosen for this comparison, because these two systems proved to be able to fulfill more or less the basic requirements in order to successfully run a job on several Grid middleware systems.

Unfortunately errors were encountered while trying each SAGA implementation. To choose an API, we took into account which system had the most adaptors implemented and with which the most tests ran successfully. The following tables compare JSAGA and JavaSAGA/JavaGAT. JavaSAGA was considered together with JavaGAT, because it mostly relies on its JavaGAT adaptor at it is assumed that if JavaGAT successfully achieves a task, also JavaSAGA should be able to do it soon, without too much modifi-

cations in the code.

Aspect	JSAGA	JavaSAGA JavaGAT
Globus security context	Yes	Yes[a]
gLite (Globus+VOMS) security context	Yes	No[b]
Using multiple security contexts automatically for a given URL	No[c]	Yes[d]
Gsiftp data protocol	Yes	Yes
SRM support	Yes[e]	No[f]
Running jobs using the WMS	Yes[g]	No[h]
Running jobs on LCG-CE(Globus Gatekeeper)	Yes[i]	Yes
Running jobs on CREAM CE	Yes[j]	No

Table 2.2: Comparison of the JSAGA and JavaSAGA capabilities

[a] If a 'gridftp' context is initialized, the program can generate the proxy, but the proxy is saved in memory, so it cannot be reused the following time, even it would be still valid. If a 'preferences' context is initialized a proxy certificate has to be generated before it can be used. grid-proxy-init command is provided with the implementation, but it isn't very convenient that invocation has to be done separately before running a program

[b] VOMS attributes can be specified. Couldn't transfer files or run jobs with a VOMS proxy certificate, but from the errors it seems that there are no problems with the certificate

[c] Throws exception if multiple contexts are available. We have to specify which context we want to use.

[d] If one adaptor fails to successfully execute a task, the next available adaptor is tried. This doesn't mean that the transfer will eventually succeed, because it can be that the necessary adaptor isn't loaded or that there is no security context with which the transfer can be executed

[e] Supported operations: make directory, copy, list, remove. In case the CREAM CE adaptor has to be used, the SRM adaptor has to be uninstalled. There is a conflict between the https and httpg protocols.

[f] States that it cannot find the path (and the same path can be found with JSAGA). With JavaGAT files can be copied with SRM; other claimed operation: delete

[g] Jobs can be submitted and status can be interrogates. OutputSandbox is set up correctly. The FileTransfer arguments for setting up the InputSandbos and/or OutputSandBox arguments need to be escaped in the command line. Sometimes it displays errors which are hard to understand (e.g. when a file name is repeated multiple times in the Input and/or Output sandbox

[h] The JDL is generated with attributes that aren't supported by the JavaGAT adaptor. Running jobs with JavaGAT works, but only with Java 1.5.

[i] Job submission works; status can be interrogated. There are some problems regarding the InputSandbox and OutputSandbox creation, thus the output files have to be obtained with jsaga-cp.sh. The RSL is incorrectly generated. This module in JSAGA won't be developed further, because the LCG-CE is being staged out. A work around for these problems would be to use the WMS adaptor and specify the queue where the job has to be submitted.

[j] Even if the JDL file is specified, the executable has to be specified in the command line. The job description file must only contain SAGA job description attributes (the idea is to enable to submit the same job description to several middleware), and these attributes can be overwritten by command line attributes for convenience.

Aspect	JSAGA	JavaSAGA - JavaGAT
Other supported middleware	Unicore, Globus WS-GRAM, Naregi	OMII GridSAM, using the JavaGAT adaptor: WS-GRAM, SGE, Unicore, Zorilla, Koala, DRMAA
Security contexts	Java keystore, SSH asymmetric key, username/password Globus legacy Globus proxy RFC 3820, VOMS proxy with MyProxy server	username/password (for FTP, SSH and SSH trilead) it uses org.globus.tools.ProxyInit, so other types of proxy certificates can be generated too - Globus proxy RFC 3820, limited globus proxy, independent globus proxy, legacy globus proxy
Data management protocols	http, https, zip, local, iRODS, SRB, sftp	using the JavaGAT adaptor: http, https, sftp trilead, ssh trilead, command line ssh, glite GUID, glite LFN, ftp
Execution management technologies	fork, SSH	using the JavaGAT adaptor: SSH, SSH trilead,fork

Table 2.3: Comparison of the JSAGA and JavaSAGA capabilities - claimed, but not tested properties

It should be mentioned that JavaGAT did a better job at generating JDLs, although all-in-all it looks like JSAGA should be preferred. With JSAGA jobs could be run with multiple Grid middleware systems, while JavaSAGA failed to comply with its own adaptors. A big advantage for JSAGA was that it could submit jobs to the WMS and to CREAM CE and SRM is supported. Although the LCG-CE support in JavaSAGA is better than in JSAGA, this couldn't compensate for the previously mentioned advantage of JSAGA. There is a way to overcome this drawback by using the WMS adaptor. This introduces an extra overhead, but it submits the jobs correctly to the LCG-CE. It has to be mentioned that this isn't such a big drawback, because the LCG-CE computing element is going to be replaced by the CREAM CE. It also seems that adding a couple of (quite simple) improvements to JSAGA would increase it's qualities and usability, although with JavaSAGA obtaining the same results would require more time and effort.

JSAGA wasn't designed to be used by a workflow engine. It is ideal for several mono-user applications running the same JSAGA installation, but using it in a multi-user environment can cause some problems. One of these seriously affected the design decisions of the developed system.

The JSAGA application has a bootstrap loader, which is a singleton class. This loader has to be accessed in order to be able to use the JSAGA factories (SessionFactory, JobFactory, ContextFactory, FileFactory, etc.). If we don't load the JSAGA engine for every client separately, then the following behavior occurs: the first client successfully sets up his/her own security context, and starts submitting jobs. The second client also send the necessary information regarding his/her own proxy certificate, but although the

JSAGA configuration file is extended, the proxy file of the first client is going to be used.[6]

To solve this problem the following question had to be answered: "When can we end up with more than one singleton instance?" Several approaches were found which can be addressed this problem. The first solution would be to have several virtual machines, because every virtual machine has a different singletons. Also different class loaders can have different singletons. The last approach would be to use a factory class, which sometimes can create several singletons. The chosen approach is explained in the next chapter.

In the next chapter we explain the design choices and how a middleware independent job submission system can be implemented for a multi-user environment.

[6] According to the JSAGA developers it should be possible to set up programmatically several security contexts with different proxy certificates, although when this was tested, problems occurred with the location of the proxy certificates. The assumption is that the user certificate and user key has to be on the same machine where the security context is created, although this isn't possible in our case. We only have a proxy certificate obtained from the MyProxy server.

Chapter 3

Grid MIddleware Independent jOb maNager - gMInION

The next generation of applications within the eScience context is expected to require even more resources to successfully achieve the goal of a research. [54] These resources are available, but different Grid middleware systems are needed to access them, thus making the user's life hard. In this case a user can be seen as an individual or as a workflow management system.

The role of Grid MIddleware Independent jOb maNager (gMInION) is to provide an abstraction layer to submit jobs to different Grid middleware systems. In chapter 2 we saw that there are several application toolkits which already tried to address this issue. One of the biggest disadvantages of these systems is that they are designed for single user mode, thus not allowing a workflow engine to use them effectively. In this section we'll elaborate the requirements of this thesis, describe the technologies used to create a system which would fulfill these requirements and finally we'll present the design and some implementation issues.

3.1 Research Questions and Design Requirements

This thesis is concerned with virtualizing Grid computing resources by creating a middleware agnostic layer. This would allow the users to access the underlying resources in an easy way. Several issues need to be addressed to achieve an abstract layer which can be used by workflow engines to submit to several Grid middleware systems. These issues include scheduling, choosing the correct submission system, creating the job description, handling the file transfer and monitoring the jobs. All these should be done in a unified way, without making the user aware of the underlying middleware infrastructure. Since different middleware systems handle security tokens and error messages in different ways, adapting these in a manner that suits our needs represents a challenge.

The following research questions were formulated to address the most important issues which need to be investigated:

- **Is it possible to virtualize several Grid middleware systems, thus submitting to these systems in a unified way?**
- **Is it possible to retrieve intermediate results and monitor the job's state,**

while the job is running, knowing only minimal information about the underlying Grid middleware systems?

- **Can a workflow management system with multiple users use a middleware agnostic layer effectively?**
- **How can resources be selected for such an abstract layer? Is it possible to obtain information regarding the available resources and to use this information to influence the submission from the workflow manager?**

Analyzing the above mentioned questions the following functional and non-functional requirements were defined.

Functional requirements

The functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform.

To address the presented research questions, job submission has to be done in a middleware agnostic way. Job submission addresses the following problems: create the job description, input and output file transfer and managing the job. In the job description the job requirements have to be specified. These include the executable script, the arguments for the executable, the environmental variables which need to be set on the worker node, the files which need to be transferred from the client machine to the worker node and back, and other requirements which specify the characteristics of the machine where the job would run. All the above mentioned operations have to be done in a unified way for all middleware systems, thus virtualizing the underlying computing resources.

Job management refers to starting a job and canceling a running job. It also provides the ability to interrogate the jobs' state. This way the user has a better control. He/She can find out if the job was already sent to a scheduling element or if the job failed. The challenge is that different middleware system expose different states of a job. To provide a middleware agnostic layer, these have to be matched and unified.

Next we'll elaborate some of the most important job description attributes which a user has to set. The most important argument is probably the executable file. This can be a simple command (like /bin/ls) or a user created script. If the executable is a script, then the script file has to be specified as input file and it has to be transferred to the resource on which the job will be run. After specifying the executable file, the arguments can be set for this executable. These arguments are the command line arguments in case a simple command was specified. If the executable is a script, then the arguments should be the arguments that the script expects. File staging also needs to be specified in the job description in a middleware agnostic way. File stage-in requests occur before the job starts execution, while stage-out requests happen after a job completes. These requests make sure that the input files are transferred from the client machine to the execution system and back. Another important argument, which can be specified in the job description, is the environmental variable on the executing system. A set of environmental variables can be specified. Before the job starts executing these are set up on the executing system, this way the executable script can use the value of these variables.

To provide more feedback to the user, it is not enough to monitor the state of a job. Sometimes intermediate result need to be obtained while running a job. Let's assume that the user runs a simulation which sets up some essential parameters on the executing host and then starts executing. He/She would probably prefer to receive some feedback to see that these parameters were correctly set up. This way he/she doesn't have to

wait until the job finishes executing to see that some initial parameters were incorrect. This mechanism would allow him/her to cancel the job, not waisting time and resources. Also error messages could be transferred, thus allowing a user to see if the received error message is fatal, or the program can still execute without too many failures. This mechanism should be implemented in a resource independent way.

The last functional requirement addresses the selection of the computing resources to which a user can submit jobs. Using a system which virtualizes the underlying resources, allows us access to a greater amount of computing and storage resources. On the other hand, this means that the information systems for all the underlying middleware systems should be interrogated to provide a better match for a given job. For the matching process the users' security tokens, the characteristics of the job and of the resources need to be taken into account. The matching process should be done automatically, without involving the user. Another requirement would be to allow the user to specify which computing resources he/she wants to use. This seems contradictory, but sometimes users are aware of where their data is located, in which case they would be capable of choosing a computing resource located close to the storage resource where the data is. Sometimes the user wants to submit multiple jobs which need to communicate with each other. In this case allowing the user to specify the computing resources would probably result in better performance, because the access policies within a site are less restrictive then between sites and the network traffic would also be reduced.

Non-functional requirements

The non-functional requirements, or system qualities, capture required properties of the system, such as performance, security, maintainability, etc. In other words, they capture how well some behavioral or structural aspect of the system should be accomplished.

First of all such a system should be flexible, meaning that it should be adaptable and extensible. The adaptability requirement specifies that the system should provide means to enable easy submission to other schedulers. This way it will be possible to cope with changing environments. It should be possible to accommodate new modules which would allow access to new middleware systems. Changes as well as enhancements should be easily made, to support future growth. Since the APIs of the supported middleware systems may change, the system should be able to accommodate these changes easily.

The interoperability requirement should be also taken into consideration. This means that the system should be easily integrated with other (workflow management) systems enabling them to access a virtualized submission system.

From the usability point of view, the system should provide an API which can be integrated easily with other applications. This means that the exposed API should be easily used by developers of the workflow manager systems, or by client who wish to use it.

The system should perform in a reliable way. It should perform according to the specifications, providing consistent failure-free software operations as long as it is used according to the description of the API.

The system should be able to protect itself from malicious access. Which means that security considerations have to be taken into account. First of all authentication and authorization can be assured using security tokens. These tokens can be used not only for the users, but also for the resources. This way it can be prevented that a resource stores the user's private data, but the resource itself isn't trustworthy. The users' security tokens authenticate the user by specifying "who he/she is", and also authorize the user, by specifying what the user is allowed to do/access. Authorization is based on a user's membership to a VO, thus allowing resources to use local policies to restrict the access

rights. Security tokens can also be used to achieve data protection. Some jobs can process sensitive data (e.g. medical records), which need to be kept private or under strict access control. Checking the security attributes data tampering can be prevented.

All this can be achieved using Grid certificates[1] and proxy certificates (Proxy certificates are explained in more detail in appendix A.).

The last non-functional requirement addresses the systems' portability. The developed application should be easily moved from one environment to another, and allowing easy deployment and installation of the system.

3.2 Design considerations

The developed system is written in Java and has a service-oriented architecture. Using a web service interface clients can create and manage their jobs without being aware of the complex nature of the underlying middleware systems. To provide decoupling from the underlying middleware systems, on the service side, a plug-in based architecture is used. This way different submission modules can be dynamically loaded.

The resulting application is a stand-alone web service, which can be used to submit independent jobs or jobs orchestrated by a workflow manager system. This section justifies the design decisions.

3.2.1 Architecture

To meet the identified requirements, a flexible, scalable, extensible and interoperable system has to be designed, which meets the functional requirements and provides the required security functionalities.

In a client-server architecture [57] model the distributed application partitions tasks between the providers of a resource or service, called servers, and the service requesters, called clients. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. As more clients join the system, less resources are available to serve each client. All data is stored on the servers, which generally have far greater security controls than most clients. Such a model for the proposed system wasn't considered a good enough approach, because this would include a single point of failure and a lot of request could easily overload the server. It wouldn't fulfill the required scalability and extensibility requirements well enough.

In peer-to-peer architectures [57], each host or instance of the program can simultaneously act as both a client and a server, and each has equivalent responsibilities and status. In peer-to-peer networks, clients provide resources, which may include bandwidth, storage space, and computing power. As nodes arrive and demand on the system increases, the total capacity of the system also increases. In such a system there is no single point of failure. The whole system is decentralized, which also means that administering such a network is hard. These network are prone to security attacks, because unsecure and

[1] Grid certificates are X.509 certificates, which are a form of electronic identification. They are used to authenticate users by making use of a Private Key Infrastructure (PKI). These certificates need to be signed by a trusted CA, which can allow users, systems and services access to the Grid resources. The most important attributes in such a certificate are: the subject distinguished name, which identifies the person, service or system; the subject's public key; the identity of the CA which signed the certificate, attesting to the authenticity of the subject distinguished name and public key; the digital signature of the named CA.

unsigned codes may allow remote access to files on a victim's computer or even compromise the entire network. Also no link in the network is reliable, which makes this hard to use in our case. We have to know where the jobs were submitted and we need to monitor the jobs' state. The machine responsible of creating the job description, starting and monitoring the job, should be reliable.

Another considered architecture is the plug-in architecture [49]. A plug-in is a set of software components that adds specific capabilities to a larger software application. If supported, plug-ins enable customizing the functionality of an application. This would allow us the desired flexibility. This type of architecture was considered suitable enough to discuss it in more detail.

Last, but not least we have to mention the Service Oriented Architecture [48, 27]. This is a flexible set of design principles used during the phases of systems development and integration. A deployed system with a service oriented architecture will provide a loosely-integrated suite of services that can be used within multiple domains. The clients are decoupled from the services. They can access the services using a well defined interface. This type of architecture would provide us the needed flexibility and would offer loose coupling between the services and the underlying infrastructure. Although providing the appropriate security levels is a challenge, it isn't impossible to achieve it. This architecture is also further explained in this section.

3.2.1.1 Plug-in architecture

A plug-in architecture, as shown in figure 3.1, is a framework that will allow a program to "look for" add-in functionality at startup, and then allow that plug-in to load and execute the required operations. Basically it is a piece of software which can change the behavior of a previously compiled application.

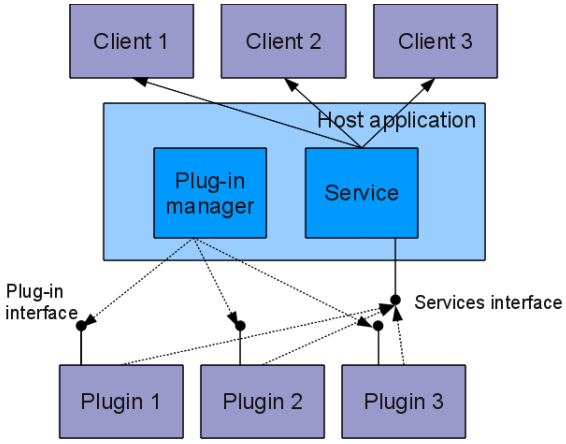


Figure 3.1: Plugin architecture. The host application provides services which the plug-in can use, including a way for plug-ins to register themselves with the host application and a protocol for the exchange of data with plug-ins. Plug-ins depend on the services provided by the host application and do not usually work by themselves. The end-users can add and update plug-ins dynamically without the need to make changes to the host application.

The principle is that the main application publishes a contract with the plug-ins in the form of an interface that the plug-ins are expected to implement in order to meet

the contract. Then the main application will scan a known location on disk (e.g. a "plugins" directory relative to its installation directory) and attempt to load all plug-ins that export implementations of this interface.

There are several benefits of a pluggable system. Such a system is considered extensible, because the application can be dynamically extended to include new features. The development of a pluggable system can be done modularly, since features can be implemented as separate components, so they can be developed in parallel. The direction of the development is clear, because the plug-in framework provides a well-defined interface.

But some of these strengths are also weaknesses. The plug-in interface needs to anticipate the ways plug-in writers want to extend the application, otherwise it restricts the extension. The provider of the plug-in framework has to make sure that the plug-in interface satisfies the intended use cases. He/She also has to manage versions and backward compatibility with existing plug-ins. Otherwise all the plug-in developers need to update their plug-ins whenever the interface changes. Although each plug-in works when tested alone, interactions between plug-ins can cause new problems, with bugs appearing only with certain combinations of plug-ins. If an existing 'client' application would like to use a plug-in architecture, it has to interact with the main framework of the plug-in architecture. This could contain libraries which aren't compatible with the libraries of the 'client' application.

The plug-in architecture is a good design choice, because it would allow developing several modules which could submit to different Grid middleware systems. However, the client has to closely interact with the main framework, which could cause problems related to the needed libraries. For example: in the case of the WS-VLAM workflow manager system, the submission module needs to be called from within a Globus container version 4.0, which is strictly bound to Axis 1.2 [56]. The problem occurs when we have a plug-in which needs to use Axis 1.4. This cannot be done, because the globus container doesn't allow the deployment of newer version of Axis.

3.2.1.2 Service Oriented Architecture

With a service oriented architecture it is possible to build reliable distributed systems that deliver functionality as services. These services comprise unassociated, loosely coupled units of functionality that have no calls to each other embedded in them. Thus using the service oriented architectural style, components are described as modular services which can be discovered and used by clients. Clients can use these services individually or they can integrate them in order to provide higher level services. This approach promotes reuse of existing functionality and dynamic compilation of existing applications. In order to achieve this, services communicate with their clients by exchanging messages, which are usually defined in terms of requests and responses (see figure 3.2).

Service providers register their services in service registries, where they advertise their capabilities and usage, in order for service consumers to be able to use them. Rather than defining an API, the service oriented architecture defines the interface in terms of protocols and functionality. An endpoint is the entry point for such a service oriented architecture implementation. Clients can use this registry in order to discover services that meet their demands. If a service presents a simple interface that abstracts away its underlying complexity, users can access independent services without knowledge of the service's platform implementation. [17]

When developing applications based on the service oriented architecture, attention has to be given to the granularity of the provided services. The larger the chunks, the fewer the interface points required to implement any given set of functionality; however, very large chunks of functionality may not prove sufficiently granular for easy reuse. The

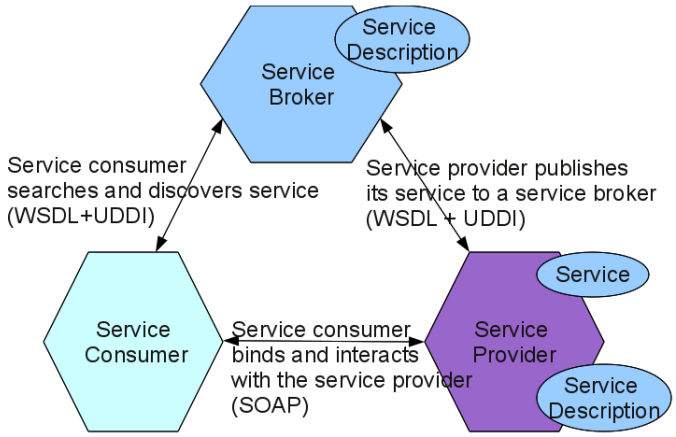


Figure 3.2: Service oriented architecture. A service provider publishes its services in a standard format in the service broker, which is accessible over the Web. A service requestor searches the registry to discover the service and its associated endpoints. It then binds with the service provider and invokes a method on the Web service using the SOAP protocol.

service oriented architecture describes loosely coupled services, which means that the participating services in such an application are unaware of each other. This provides the following benefits:

- Flexibility: A service may be deployed in any machine and relocated if necessary. As long as that service updates its registry entry, clients will be able to discover it.
- Scalability: Services can be added or removed in order to meet the demands of the application.
- Decoupling implementation from the exposed interface: If services maintain a standard interface, new or updated implementations can be introduced without disrupting the functionality of any application.
- Fault tolerance: If a service becomes unavailable for any reason, clients can still query the registry for alternate services that offer the required functionality.
- Reusability: A service can be part of more than one application, as it can interact with multiple clients. This means that a lot of redundancy can be eliminated.
- Interoperability among different systems and programming languages that provides the basis for integration between applications on different platforms through a communication protocol.

Some of the disadvantages of such a service oriented architecture are the following:

- Security : trust has to be established between the service consumer and the service provider, this means that security measures need to be provided to restrict access to services.
- Maintenance: Bugs may be difficult to find. It is difficult to modify the exposed interface if services are deployed to different locations.

Considering the above mentioned features, a service oriented architecture is preferred because of its low coupling and interoperability characteristics. This architectural style also enables the integration of application which use different library versions. Section 3.2.3 describes how the security disadvantage was handled.

To maintain the extensibility offered by a plug-in architecture on the server side, this style was used. Thus the final architectural design is a combination of a service oriented architecture and a plug-in architecture. The client can access the system's functionality using web service calls, while in the background the service dynamically loads the modules which actually execute the required operations.

3.2.2 Design patterns

Design patterns are used to solve a particular problem in the architecture. The list of design pattern described by Gamma et. al [34] is recognized as the current standard in design pattern classification. This list contains over 20 design patterns, which can be divided into creational patterns, structural patterns and behavioral patterns. For the purpose of this thesis it is not necessary to introduce all of them. Only those are described in more detail which were used to develop the system.

As already stated on the service side we have a plug-in architecture. This can be achieved using the Factory design pattern or the Abstract Factory design pattern. These are two very similar design patterns. Both are creational patterns, meaning that they abstract the object instantiation process. They help make the overall system independent of how its objects are created and composed, by hiding how the objects are created. The difference is that the Factory method is a class creational patterns, focusing on the use of inheritance to decide which object should be instantiated, while the Abstract Factory method is an object creational patterns, focusing on the delegation of the instantiation to another object.

The Factory design pattern defers the creation of an object to its subclasses. It defines an interface for creating an object, but lets the subclasses decide which class to instantiate. This can be done at compile time (static or class scope) or at run time (dynamic or object scope). This makes it easy to switch between different implementations, thus allowing a higher level of decoupling. The code is made more flexible and reusable by eliminating the instantiation of application specific classes.

The Abstract Factory pattern is very similar to the Factory Method pattern. One difference between the two is that with the Abstract Factory pattern, a class delegates the responsibility of object instantiation to another object via composition. This pattern only exposes the product interfaces to client code. Factory classes are exposed to product implementations, however, since factories only create products there are no serious logical dependencies.

As shown in figure 3.3 the abstract classes for the related products and the abstract factory for creating the different instances of the abstract products have to be defined. Then these the abstract classes and the abstract factory has to be implemented. The factories are responsible of the creation of the related products and each group of related products is represented by a Implementation.

This design pattern should be used when a class can't anticipate the class of objects it must create or when a system must use just one of a set of families of products. It can also be used when a constraint needs to be enforced regarding to a family of related product objects which need to be used together. This method isolates clients from concrete implementation classes and enforces the use of products only from one family. In our case this is desired, because it wouldn't make sense to create part of the job description using an instance which could submit to local resources, while the job

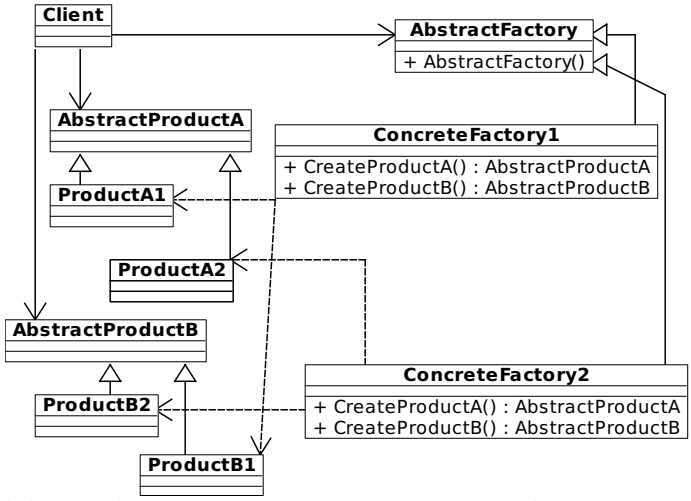
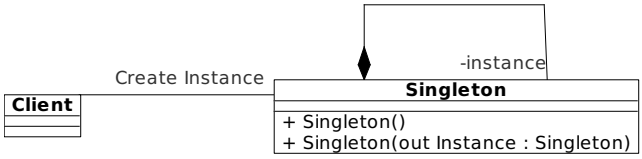


Figure 3.3: The Abstract Factory design pattern. The AbstractFactory defines a factory type that list operations for creating the set of related abstract product types. The ConcreteFactory implements the list of operations and create the concrete product objects that are associated with the specific concrete factory class. The AbstractProduct declares the operations available for the specific product type. The implementation of the product type is delegated to the subclasses of the product type. The ConcreteProduct is created by a specific concrete factory object and is the realization of a specific product type. The Client object is decoupled from the ConcreteFactory and ConcreteProduct objects and work with the interfaces declared by the AbstractFactory and AbstractProduct types.

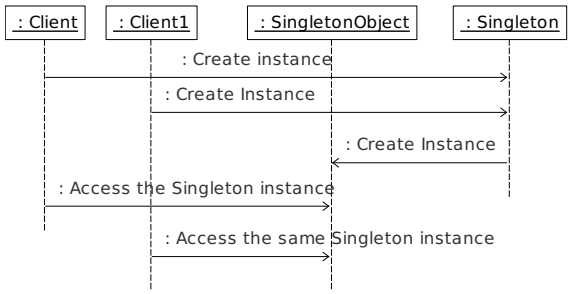
would be submitted to a cluster.

Using this pattern abstracts away the creation process of the components from the code and it allows these to be replace by some other implementation of the Abstract Product. Flexibility and loose coupling is gained. Another advantage is the reusablilty of the same code that uses the Abstract Products.

In our case choosing the abstract factory design pattern can be useful, especially if we take into account the class loading in JSAGA. Using the abstract factory design pattern combined with dynamic class loading would allow us to use several JSAGA instances (one for each user) and it would also allow the creation of other modules which could submit to different underlying systems.



(a) Class Diagram



(b) Sequence Diagram

Figure 3.4: *Singleton design pattern* In the Singleton Pattern we have at any given instance a single instance of the Singleton Class active. All instantiation of the Singleton Class references the same instance of the Class. Singletons maintain a static reference to the sole singleton instance and return a reference to that instance from a static method.

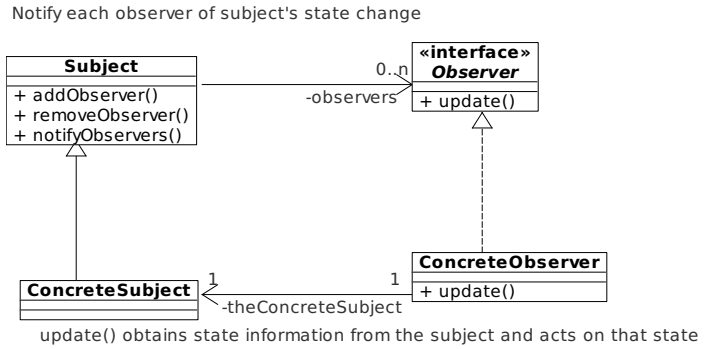
The next presented design pattern is the Singleton pattern. This assures that the accessed class has only one, globally accessible instance, as shown in figure 3.4. This is useful when exactly one object is needed to coordinate actions across the system.

The singleton pattern is implemented by creating a class with a method that creates a new instance of the object if one does not exist. If an instance already exists, it simply returns a reference to that object. To make sure that the object cannot be instantiated any other way, the constructor is made either private or protected. The class also contains a static instance of itself. The access to the static instance is provide via a static method.

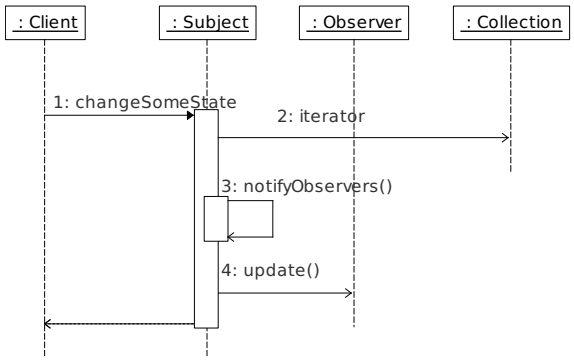
This design pattern was chosen, because web services are stateless, but a client which accesses the designed web service has to be able to create several job descriptions and access his/her credential context which is set up only once (at least while the delegated proxy is valid).

The last presented design pattern is the observer pattern. This pattern defines a one-to-many dependency between collaborating objects. As shown in figure 3.5, it enables the partitioning of a system into observers that react when their subjects change state. It basically decouples the event source from the event monitors. The observer pattern can be used when a client wants to bind the standard output and error file, obtained while a job is running, to a GUI. Every time new data is received from a worker node, the GUI has to be notified to update its interface. The only thing that needs to be done is to create a thread which periodically checks if the content of a file changed. If it does, it sends a notification to all objects which subscribed to watch its state.

Unfortunately this pattern cannot be used to monitor the status changes of a running job, because these don't send notifications when a job changes its status. We have to 'pull' this information from the underlying middleware system. To virtualize the underlying system, the client has to request this information from the gMInION service.



(a) Class Diagram



(b) Sequence Diagram

Figure 3.5: *Observer design pattern* The subject, which is an event source, maintains a collection of observers and provides methods to add and remove observers from that collection. The subject also implements a notifyObservers() method that notifies each registered observer about events that interest the observer. Subjects notify observers by invoking the observer's update() method. In 3.5(b) some unrelated object will invoke a subject's method that modifies the subject's state. When that happens, the subject invokes its own notifyObserver() method, which iterates over the collection of observers, calling each observer's update() method.

3.2.3 Security

One of the non-functional requirements specified in section 3.1 was concerning the security of the developed application. It is desired that the communication between the client and the server takes place over a secured connection. This requires the server and the client to exchange certificates, thus accomplishing mutual authentication.

The second security aspect concerns the management of the proxy certificates required to run a job on the underlying Grid middleware. A secured connection is required for one of the the functional requirements, namely: obtaining partial results from a running job, This is set up differently then the secured connection mentioned in the first paragraph of this section, because this connection has to be established between the client and the node where the job is effectively running. This is realized using Openssl s_server and s_client, which are also shortly described in this section. [63]

3.2.3.1 Mutual authentication

When using mutual authentication the data being sent is encrypted by one side, transmitted, then decrypted by the other side prior to any processing. This is a two-way process, meaning that both the server and the browser encrypt all traffic before sending out data. With mutual authentication, the actual entity's certificate or an entity in the certificate chain (thus the descendant of the entity's certificate) has to be trusted. When messages are sent with mutual authentication, a connection is possible only if the client trusts the server's certificate and the server trusts the client's certificate. The process of exchanging certificates and setting up connection properties is called the Secure Sockets Layer (SSL) [57] handshake.

Secure Socket Layer (SSL) is based on a public key cryptography system, in which separate keys are used for encryption and decryption. To assure authorization, data integrity and confidentiality an SSL session has to be set up before an HTTP transaction takes place.

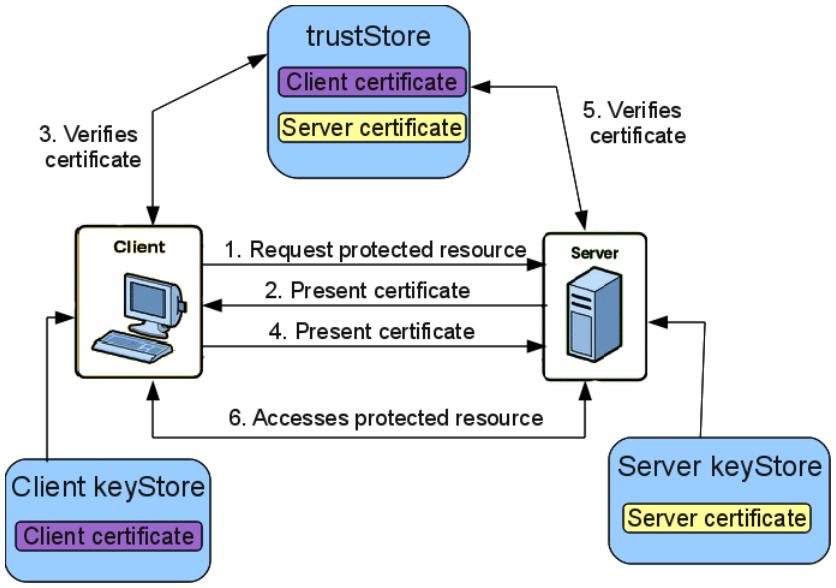


Figure 3.6: *Mutual authentication* Mutual authentication refers to a client or user authenticating themselves to a server and the server authenticating itself to the user in such a way that both parties are assured of the others' identity.

Figure 3.6 shown how mutual authentication can be achieved. Both the server and the client has to trust each others certificate. A certificate is trusted, if it is valid and it is signed by a trusted CA. In the figure we can't see the details of the communication, which include agreeing on the used protocol and send verification messages to make sure that the symmetric keys, which is going to be used to encrypt the messages, is received

correctly.[2]

Using mutual authentication with SSL provides authentication, confidentiality and integrity. Authentication is addressed, because during the initial attempt to communicate with a server over a secure connection, that server will present a set of credentials in the form of a server certificate. The purpose of the certificate is to verify that the site is who and what it claims to be. The server may also request a certificate from the client, which states who the client claims to be (which is known as client authentication).

Data confidentiality is provided, because SSL responses are encrypted so that the data cannot be deciphered by a third party and the data remains confidential. SSL also helps guarantee that the data will not be modified in transit by that third party, which means that data integrity is addressed.

Implementing SSL has one disadvantage: it requires both parties of the communication to do extra work in exchanging handshakes and encrypting and decrypting the messages. These operations are more CPU-intensive then unencrypted communication, thus increasing the load of the systems and making this form of communication slower than communication without SSL.

To see how a Tomcat server can be configured to require mutual authentication using the existing X.509 certificates, please refer to appendix C.

3.2.3.2 OpenSSL

OpenSSL [63] is an open-source implementation of the SSL and Transport Layer Security protocol and related cryptography standards.

With OpenSSL private and public keys can be created and managed, X.509 certificates and CRLs can be created and validated against the installed CA certificates, proxy certificates can be validate, Messages Digests can be calculated, ciphers can be encrypted and decrypted, etc.

Using proxy certificates and openssl s_client mutual authentication can be established between the client and the server side, and even more, on the established connection files can be sent over to the listening server. Basically server- and client side certificates have to be set up, and then using openssl s_server the host starts to listen on a given port. Meanwhile the client can send the content of a file using openssl s_client.

3.2.3.3 Managing proxy certificates

Users of Grid systems wish to maintain the privacy of their documents, although they are accessing resources across multiple administrative domains,whilst they also require easy access to these resources. Security technologies which involve one single authority for

[2]When the client wants to access a secured resource, it has to send the server a message containing the SSL version and the cipher suites the client can talk. The client sends its maximum key length details too. The servers' return message specifies the version of SSL and the ciphers and key lengths to be used in the conversation, chosen from the list offered in the client's message. The server also flags if it requires a client certificate. Then the server and the client inspect each others certificates. If both certificates are considered valid, the client generates a symmetric key and encrypts it using the server's public key (certificate). Then it sends this message to the server. The client then sends a Certificate verify message in which it encrypts a known piece of plain text using its private key. The server uses the client certificate to decrypt, therefore ascertaining that the client has the private key. After this the client sends a cipher specification message telling the server that all future communication should be with the new key. The client now sends a Finished message using the new key to determine if the server is able to decrypt the message and the negotiation was successful. The server sends a Change Cipher Specification message telling the client that all future communications will be encrypted. The server sends its own Finished message encrypted using the key. If the client can read this message then the negotiation is successfully completed.

managing (generating, storing, ...) credentials cannot provide the indispensable requirements for achieving successful authorization and authentication for the Grid community. Delegation is needed, so a server can access Grid resources on the user's behalf.

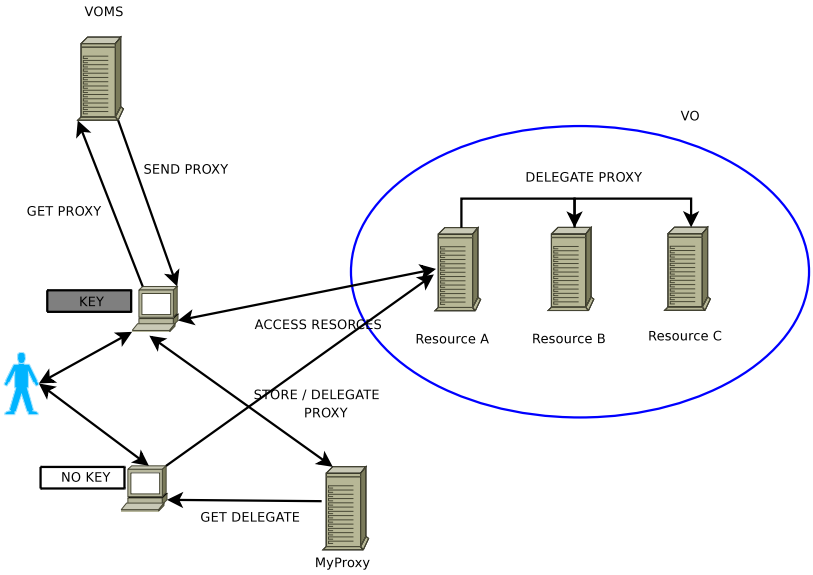
Considering the proxy delegation problem a choice had to be made between the delegation service in the GT and the MyProxy service. The GT delegation service accepts a credential from the user and provides access to that credential to any authorized service that runs in the same container. Upon delegation to the service an endpoint reference to the delegated credential is returned to the client, which can then be sent to other services as a handle to the credential. This service couldn't be used, because only the delegation credentials could only be used by services hosted in the same container as the delegation service.

Thus in the developed application the MyProxy service is used, which is described in the following paragraphs.

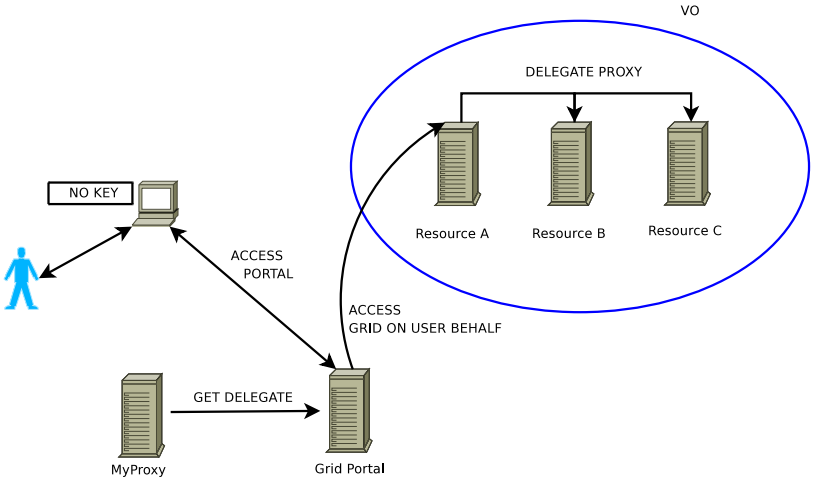
MyProxy

The MyProxy service was originally intended to authenticate users on web portals [47]. Typically users would enter their password on the web portal web interface which in turn will contact the MyProxy server and delegate proxy certificates on behalf of the user. MyProxy's potential led further development to extend the service and support more feature most noteworthy are mobility and renewal[47].

The MyProxy service is an on-line credential repository system and it's included in GT4. Grid users in possession of a certificate can upload their long lived certificate to this server and manage their certificates on-line. It must be noted that such a server containing so much sensitive data would generally be well maintained and secured by Grid administrators [46]. Part of their activity is monitoring the system, detecting intruders and defining access control lists. The MyProxy service acts as a delegation broker, after authenticated, users will get a delegation of their certificate from the MyProxy service hence no private keys are transmitted into the open.



(a) Store and retrieve



(b) Retrieve

Figure 3.7: Credential management with MyProxy. MyProxy is a client-server system, where clients can store credentials in an online repository for later retrieval. MyProxy uses X.509 proxy certificates to support storing and retrieving credentials without exporting private keys.

The MyProxy service is not a substitute for GSI but rather a complement to the infrastructure. It aims at facilitating the usage of certificates on Grids in three main areas, web portals, mobility and renewal. Since web browsers and HTTP protocol are not capable of authenticating a Grid user, the idea of portals became popular as a way to work around these authentication problems. These communicate with the MyProxy services to get proxy delegates and access the Grid services on the users behalf.

With the MyProxy service also mobility is made much simpler. Users need not have a local copy of their long lived credentials instead they can sit in-front of a Grid enabled

computer anywhere in the world and contact the MyProxy service to delegate a proxy certificate.

These users don't have to carry their private keys, they can authenticate with a user identity and a pass phrase chosen by themselves, which conform to several security policies. To enhance the security of the MyProxy server, these pass phrases are used to encrypt the users credentials. Storing the user credential on MyProxy is presented on the Figure 3.7(a).

This step requires the private key to be available on the machine from where the user accesses the VOMS. After the proxy is retrieved it can be stored on MyProxy. Figure 3.7(b) outlines how a stored credential can be retrieved.

Proxy certificates are generated with a short lifespan, generally a few hours, to prevent . This criteria poses a greater problem as to what will happen when proxies expire whilst jobs are still running or queued on some computing element. Also certificates could expire whilst transferring large amount of data. A naive solution would be to increase the life time of proxy certificates which in itself beats the scope of having proxy certificates in the first place. Still it is impossible to know beforehand how long a job will take to complete this is due to the fact that a job could be stuck on a job queue for an indefinite amount of time.

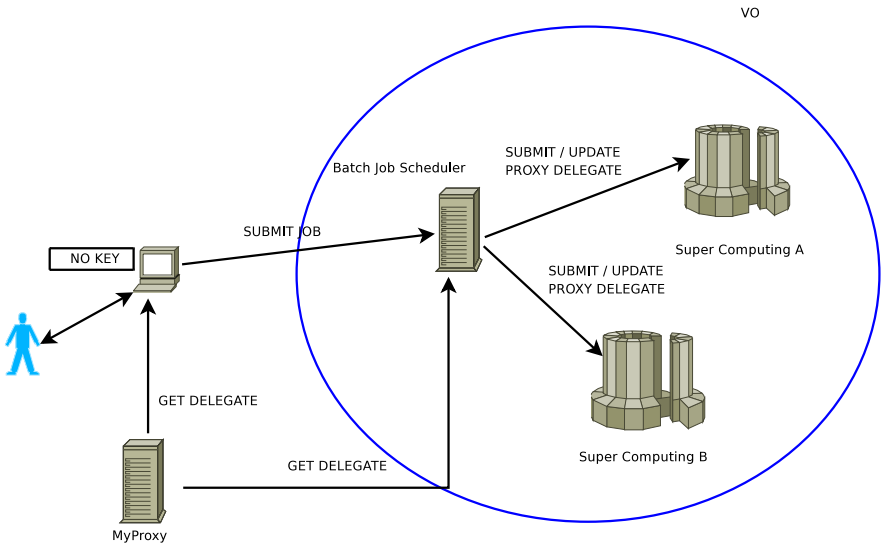


Figure 3.8: MyProxy Renewal. Authorized clients holding valid credentials nearing expiration are allowed to authenticate to the MyProxy server and "renew" those credentials, by obtaining a new proxy credential with the same certificate subject.

The WMS addresses this problem using MyProxy. Upon job submision to the WMS the users certificate is delegated to the WMS. This delegation is the normal process for job submission and has nothing to do with MyProxy. With the chaining effect of delegation the WMS will re-delegate the proxy to other resources so as to submit the job. If the user wishes to make use of the renewal service, it must be explicitly specified in the JDL file by specifying the MyProxy server hostname. With the renewal option enabled the WMS contacts the renewal service to register the delegated proxy for renewal. The renewal service is then responsible to keep in mind when the proxy should be renewed. Upon approximation to the proxy expiry time, the renewal service will contact the MyProxy

server to get a new delegation. Once retrieved the proxy certificate is propagated to the computing resources. This enables jobs to have a valid certificate throughout their lifetime without compromising the lifespan of proxy certificates.

3.3 Proposed architecture

We propose an additional layer between the client or workflow engine and the underlying middleware systems. This layer would virtualize the underlying resources and would provide a unified access point to several Grid middleware systems. The gMInION layer is illustrated in figure 3.9. The layer is responsible of setting up the necessary security contexts, choosing the middleware which should be used to submit the jobs and managing these jobs.

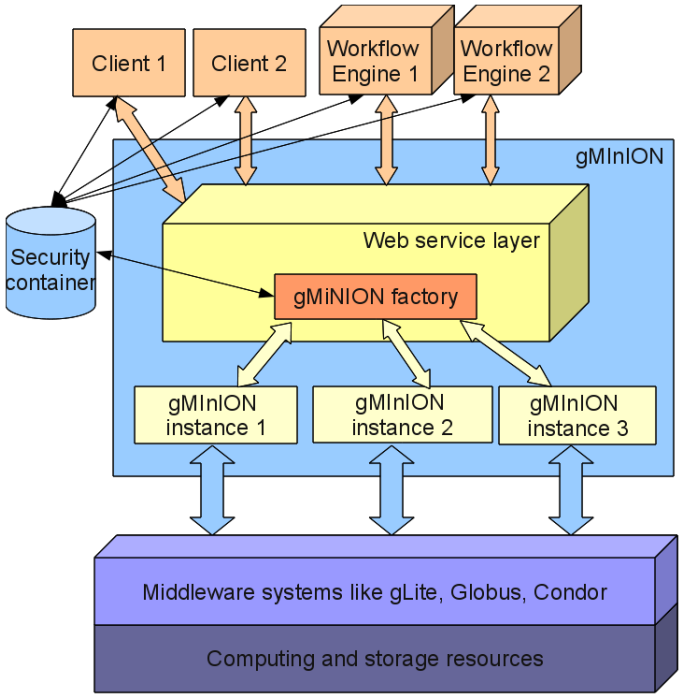


Figure 3.9: Proposed architecture. The client accesses the gMInION layer using web service calls. The gMInION layer is responsible of setting up the required security context, choosing a suitable middleware and monitoring the jobs.

The proposed architecture is a combination between a service oriented architecture and a plug-in architecture. Clients need to delegate their security tokens to a security container. Clients access the system using web service calls. On the service provider side a gMInION factory singleton object is available. This evaluates, based in the available user's security tokens, what type of middleware system can be used. The gMiNION factory can be seen as a concrete factory in the abstract factory design pattern. The gMInION factory dynamically loads the necessary plug-in, and creates a gMInION instance which will interact with the underlying middleware system.

Using the plug-in architecture allow us to easily add new components, which could address other underlying infrastructures. As can be seen in figure 3.9, the clients don't

interact with the middleware systems directly. This makes our approach middleware agnostic and allows the client to use the virtualized underlying resources as one entity.

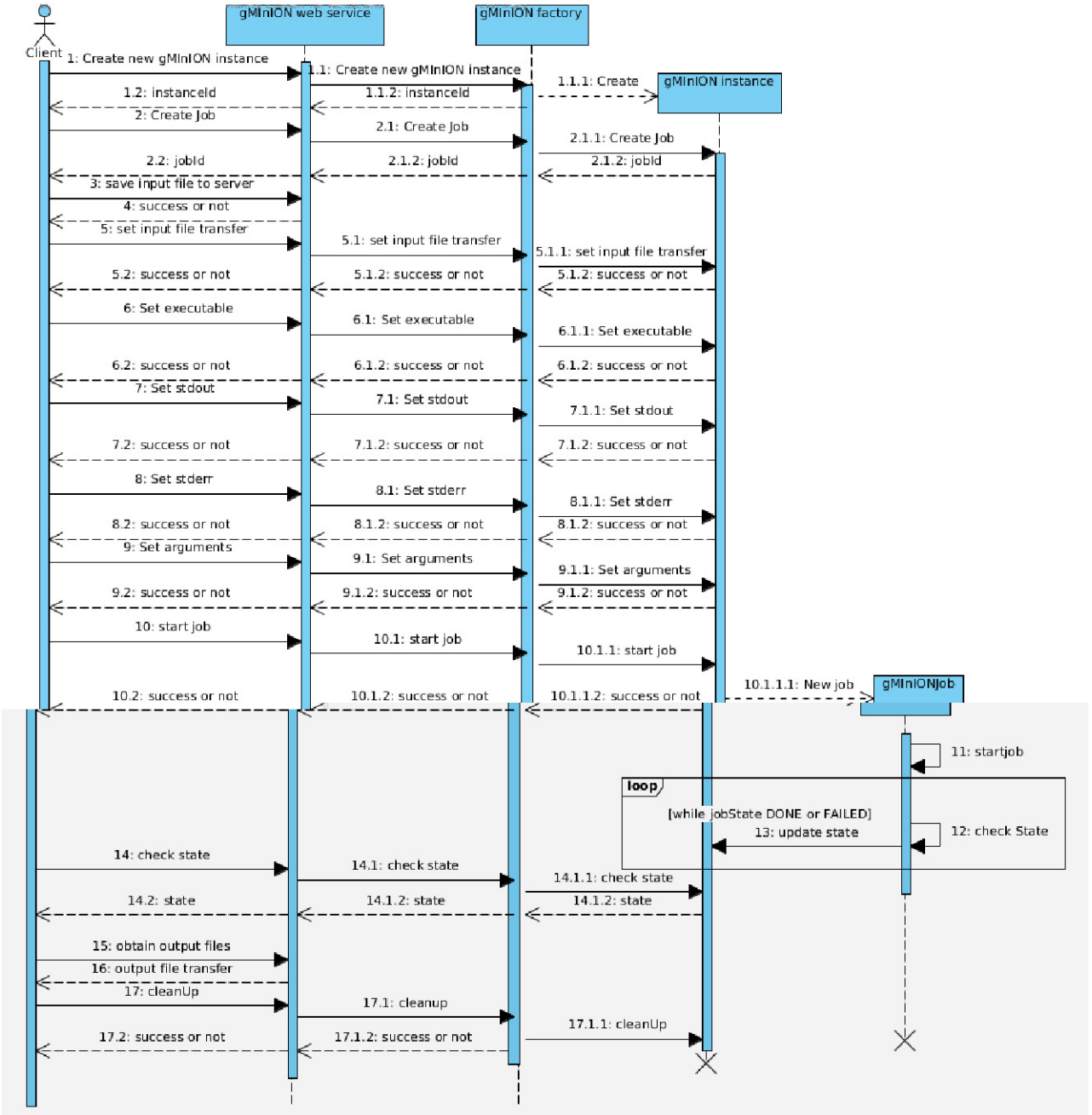


Figure 3.10: Sequence diagram for submitting a job. This figure shows the interaction between the participating objects when a user wants to create and manage a job.

Figure 3.10 shows the message exchange which needs to be performed when a job is

created. The client initiates the process by asking for a new gMInION instance. In this step the security context is set up for the client. After this step the client can create a job and set up the job description. In figure 3.10, in the job description some input files, the executable, the standard error, the standard output and the arguments of the executables are specified. Then the job is started. The client periodically checks the state of the job. When the job execution finishes, the client obtains the output file, then cleans up the gMInION instance.

Running a script requires basically the same calls. The user only has to make sure that the required script is specified as an input file. We propose to develop a client side module, which would allow easier interaction with the system. One of the advantages that this module would provide, is that it would make sure that the executable script will be specified as an input file and that this script will be automatically transferred to the web service.

Interaction between a client and the gMiNION layer

This section focuses on the interaction between the client and the developed system. The operations of the system are exposed as web services. The client has to use standard web service invocation methods to access these services. If the reader is familiar with these method, he/she is invited to skip this part and continue reading the implementation details in section 3.4.

Web services constitute a standard way to realize a service oriented architecture. Web services are a distributed computing technology, based on the principles of the service oriented architecture that make up the building blocks of developing loosely coupled distributed applications. The World Wide Web Consortium (W3C) defines them as [39]:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards*

Graphically a web service invocation could be represented as presented in figure 3.11.

SOAP is an XML-based protocol for exchanging structured information over decentralized, distributed environments. The most commonly used protocol to access web services is HTTP(S).

In order to construct a SOAP message which would invoke the desired web service, a description of the service is needed. This is accomplished through the WSDL document, which describes the public interface of a web service. WSDL describes web services as collections of network endpoints, or ports. The abstract definitions of ports and messages are separated from their concrete use or instance, allowing the reuse of these definitions. Messages are abstract descriptions of the data being exchanged, and port types are abstract collections of supported operations.

At an abstract level, a WSDL document describes a web service in terms of messages it sends and receives, and the data types used in these messages. The description of these data types found in WSDL are provided by an XML Schema [28]. Additionally at the abstract level, the WSDL defines the service ports.

At the concrete level, the WSDL defines the protocol and data format specifications for particular port types. Here the operations and messages are bound to a concrete network protocol and message format.

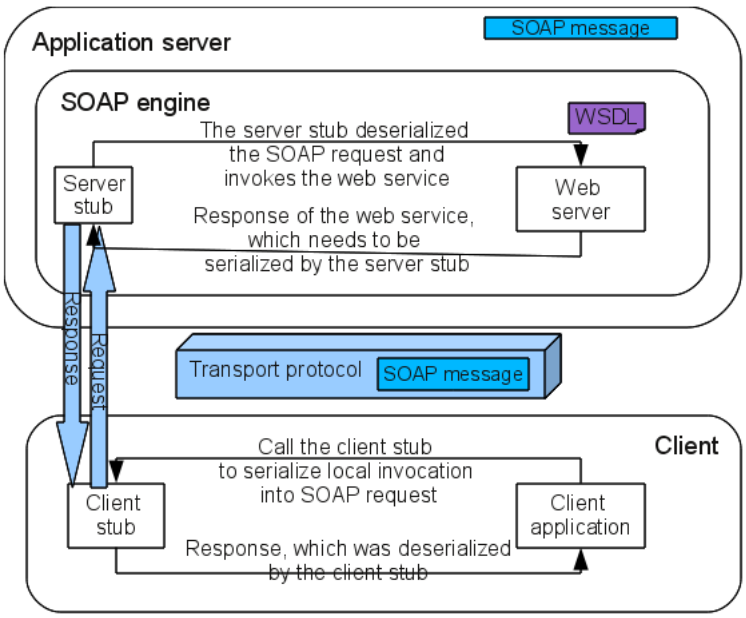


Figure 3.11: Web service invocation. When the client application needs to call the web service, it will perform a local call to the client stub. The client and server stubs are responsible of the marshalling/unmarshalling of the SOAP requests and responses. Only SOAP messages are transferred between the client and server. The server can carry out the work it's been asked to do, only after the SOAP call is transformed into something it can understand.

So after locating the web service and obtaining the WSDL file, a client can generate a stub file, which would dynamically generate SOAP requests and interpret SOAP responses. It is enough if the stub is generated only once, unless the WSDL interface of the service changes. After this step the methods in the stub class can be invoked and the client doesn't have to worry about SOAP messages.

Whenever the client application needs to invoke the web service, it will call the client stub[3]. The client stub will turn this 'local invocation' into a proper SOAP request. This is often called the marshaling or serializing process. The SOAP request is sent over a network using the HTTP protocol. The server receives the SOAP requests and hands it to the server stub. The server stub[4] will convert the SOAP request into something the service implementation can understand (this is usually called unmarshaling or deserializing). Once the SOAP request has been deserialized, the server stub invokes the service implementation, which then carries out the work it has been asked to do. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand. Finally the application receives the result of the web service invocation.

On the server side, web services are usually deployed within application servers that

[3]A client stub is responsible for conversion of parameters used in a function call and deconversion of results passed from the server after execution of the function.

[4]A server stub is responsible for deconversion of parameters passed by the client and conversion of the results after the execution of the function. It makes the development of the server side code easier.

use some communication protocol for sending and receiving requests and responses. Basically the application server provides a 'living space' for applications that must be accessed by different clients. The SOAP engine runs as an application inside the application server. The SOAP engine knows how to handle SOAP requests and responses. It instantiates requests from the SOAP message and invokes the desired web service, which generates a response for the SOAP engine. The SOAP engine generates a SOAP messages and feeds it back to the application server, which transfers it back to the client. In practice, it is more common to use a generic SOAP engine than to actually generate server stubs for each individual web service (client stubs are still needed for the client). One good example of a SOAP engine is Apache Axis.

Next we summarize the advantages and disadvantages of web services. These services are platform-independent and language-independent, since they use standard XML languages. This means that the client program can be programmed in C++ and running under Windows, while the web service is programmed in Java and running under Linux. Web services also provide a loosely coupled system, in which the client might have no prior knowledge of the web service until it actually invokes it. The messages are transmitted mostly over HTTP. This is a major advantage since most of the Internet's proxies and firewalls won't restrict the HTTP traffic.

Although transmitting all the data in XML is obviously introduces overhead, because it is not as efficient as using a proprietary binary code. What is won in portability, it's lost in efficiency. Another disadvantage comes from the fact that HTTP and HTTPS are "stateless" protocols. The interaction between the server and client is typically brief and when there is no data being exchanged, the server and client have no knowledge of each other. More specifically, if a client makes a request to the server, receives some information, and then immediately crashes due to a power outage, the server never knows that the client is no longer active. The server needs a way to keep track of what a client is doing and also to determine when a client is no longer active. Typically, a server sends some kind of session identification to the client when the client first accesses the server. The client then uses this identification when it makes further requests to the server. This enables the server to recall any information it has about the client. A server must usually rely on a timeout mechanism to determine that a client is no longer active. If a server doesn't receive a request from a client after a predetermined amount of time, it assumes that the client is inactive and removes any client information it was keeping.

3.4 Implementation details

Using a web service interface clients can create and manage their jobs without being aware of the complex nature of the underlying middleware systems. Analyzing the user credentials, different security contexts are set up, which assure that the user can access the resources he/she wants to use. Usually this step should be done only once, and then the client should be able to create several job descriptions and submit the corresponding jobs.

In the job description several properties can be specified ranging from the ones that are mandatory (like the executable file) to the ones which create more sophisticated jobs, like: arguments of the executable, environmental variables on the node where the job will run, the queue to which the job should be scheduled, etc. Standard output and error files can be redirected and the working directory on the worker node can be set.

To obtain a system which can submit fulfill the specifications four modules were developed. These are the following:

- gMInION_Factory

- gMInION_Instance
- gMInION_client
- QueryBDII

First we'll describe the web service and then we'll elaborate the above mentioned modules.

- start_job : This method submits the job corresponding to the job identifier.
- cancel_job : This method cancels the corresponding job.
- setEnvironment : Extra parameter: a vector of StoreValues objects. This construct helps on serializing/deserializing value pairs. Using this method allows us to set up environmental variable on the machine where the job will be executed.
- getSubmittedFile : This method returns all the arguments describing the job, which were set before the job was submitted. It can be used only after the job is submitted.
- setQueue : Extra parameter: the queue name where we wish to submit the job. This way the job scheduling can be influenced.
- setExecutable : Extra parameter: the executable, as is a simple unix command like "/bin/ls"
- runScript : Extra parameter: the script name which has to be run, this method set the executable to "/bin/sh", transfers the specified script and sets the arguments of the executable to run the script
- setArguments : Extra parameter: a list of arguments for the executable
- setCPUTime : Extra parameter: the estimated total number of CPU seconds which the job will require
- setStdOut : Extra parameter: a string specifying the file name where the standard output stream should be redirected
- setStdErr : Extra parameter: a string specifying the file name where the standard error stream should be redirected
- inputFileTransfer : Extra parameters: string specifying the source file and another string specifying the destination file. This method is used to set up the input files in the job description.
- outputFileTransfer : Extra parameters: string specifying the source file and another string specifying the destination file. This method is used to set up the output files in the job description.
- setWorkingDir : Extra parameter: a string specifying the working directory for the job on the machine where it's going to be executed
- getFileToSubmit : This method returns the previously set attributes for a job. The job doesn't have to be submitted. This can be used to see if a given attribute is set or not and if it has the correct value.
- getState : Obtains the state of the submitted job.
- tryToGetErrorMsg : In case the job fails to execute successfully, this method tries to obtain the reason why the job couldn't be executed.
- sendFile : This method is used for file management. It takes two extra parameters: a string which specifies the source file and another string which specifies the destination file. The source and destination file strings need to specify the protocol which has to be used to transfer the file (e.g. gsiftp://, file://, srm://).

Apache Axis [56] was used to create the client side stubs using the web services' WSDL definitions.. Axis is a SOAP engine, a framework to construct SOAP processors such as

clients and servers. Axis also has built-in support for WSDLȦxis is an open source tool which offers a simple, well tested, and powerful solution to a complex problem.

Axis can run inside a container, such as Tomcat [14]. Tomcat is the official reference implementation JavaServlet Pages and Servlet containers. Tomcat is able to keep objects in memory. Axis can work with any valid servlet container, but it is frequently teamed with Tomcat.

Although there are more application servers which provide extra functionality like: load balancing, fault tolerance, transactions, resource pooling, caching, process management, Tomcat was chosen because it is open source, lightweight, stable and well tested. Another reason why Tomcat was chosen is because it can be easily configured. The data should be transmitted over a secured connection and mutual authentication should be implemented, preferably using the available Grid proxy certificates. As we'll see later in section C these requirements can be easily achieved in Tomcat.

To use Axis with Tomcat an axis directory has to be created in Tomcat's 'webapps' directory. This should contain the necessary axis libraries.

3.4.2 Developed modules

gMInION_Instance
This module is responsible of interacting with the MyProxy server to obtain the user's delegated credential. These credentials are needed to set up the user's security contexts. One user can run several jobs using the same security context. This module has to keep track of all the jobs that were submitted. It also has to be aware of jobs that weren't submitted yet, but which already obtained a job identifier while their job description is being created. All the above mentioned jobs have to belong to the same user.

gMInION_Instance uses the API offered by JSAGA [53]. It creates and configures the *jsaga-universe.xml* file, enabling several security contexts depending on the underlying Grid middleware where the jobs will run. It also creates the session and the job description file which are used to submit a job using JSAGA.

In a separate thread the state of the submitted job is periodically interrogated. This way when the job is done, the client can be notified and the files specified in the output sandbox can be obtained automatically.

This package contains the *ManageCredentialsI* and the *ManageJobsI* interfaces. If these interfaces are re-implemented using other APIs, then the abstract layer would be able to submit jobs to other infrastructures too.

gMInION_Factory
The gMInION_Factory module is used to manage the gMInION_Instances. It basically keeps a map of instance identifiers and instances. In this way it assures that for each user the corresponding instance is used.

The client has to transfer some information to access the client's delegated proxy credential on the MyProxy server, and then with an identifier received from the server, he/she should be able to specify which gMInION_Instance should be used.

Whenever a client wants to set up a new security context, a new gMInION_Instance is created. Each time the gMInION_Instance library and the JSAGA libraries (engine and adaptors) are dynamically loaded. The following code snippet presents how the dynamic class loading is done:

```
ClassLoader prevCl =
  Thread.currentThread().getContextClassLoader();
URL[] urlL = urlfromDir(new File(url));
```

```
URLClassLoader urlCl = URLClassLoader.newInstance(urlL, prevCl);
String classNameToBeLoaded = "nl.submitter.jsaga.gMInION";
```

The urlfromDir method returns a list of URL, which contains an URL entry for every file in the directory. When loading classes a class loader will check if the class was already loaded. If it wasn't then it will ask the parent class loader to load the class. If the parent class loader cannot load the class, it attempts to load the class itself. But our application is running in a Tomcat container, so we have to look at the class loading in Tomcat, which is a bit different. When a request to load a class from the web application's class loader is processed, this class loader will look in the local repositories first, instead of delegating before looking.

When Tomcat is started, it creates a set of class loaders that are organized into the following parent-child relationships, where the parent class loader is above the child class loader: The common class loader contains additional classes that are made visible to both

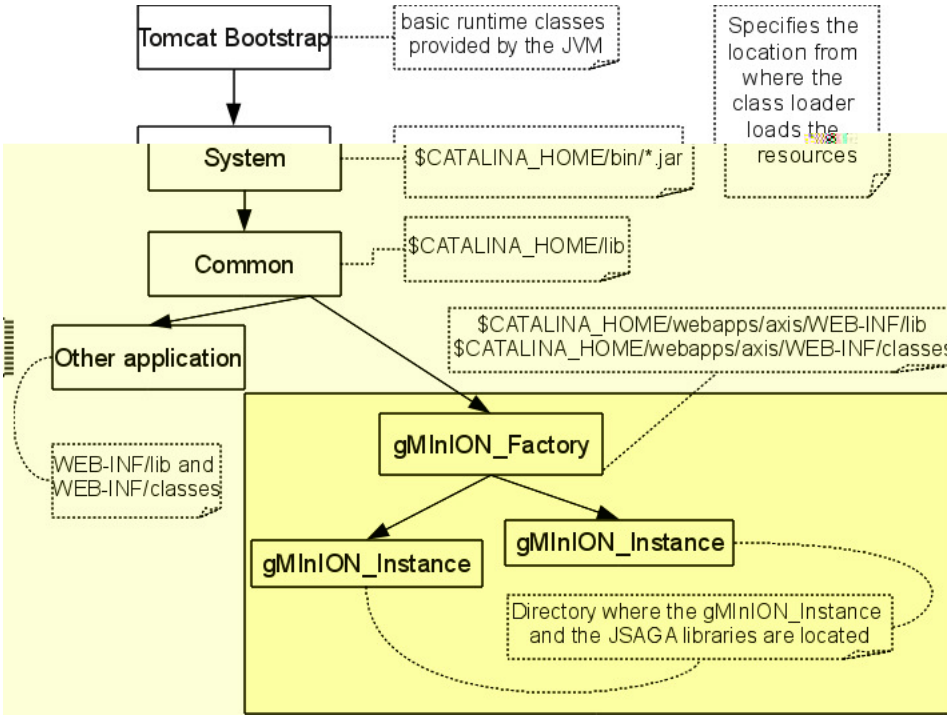


Figure 3.12: Tomcat class loader hierarchy. This emphasizes how the necessary classes are loaded. The highlighted part shows that the gMInION factory classes are located in the web application's library in Tomcat, while the gMiNION instance classes are loaded from a different location.

Tomcat internal classes and to all web applications. All unpacked classes and resources in "lib" directory located in the Tomcat home directory, as well as classes and resources in JAR files are made visible through this class loader.

From the perspective of a web application, class or resource loading looks in the following repositories, in this order:Bootstrap classes of the JVM, system class loader classes, the classes and library file located in the directory of the web application, the libraries located in the Tomcat home directory.

Because the gMInION_Factory modules parent class loader is the common loader, the

parent class loader won't be able to load the gMInION_Instance, because these libraries (gMInION_Instance and the JSAGA libraries) aren't stored in the "lib" directory of the web application.

After loading the necessary classes, Java reflection is used. Java's Reflection API's [18] makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection.

A typical way of using the reflection API involves the following steps:

1. get a Class object representing the class with a particular name;
2. from that Class, request a Method object representing the method with a particular name and parameter types;
3. call the method in question via the Method object.

Using dynamic class loading and the reflection API enables a pluggable architecture. In case another implementation of gMInION_Instance is created, probably accessing other underlying middleware resources, this can be easily used. We only need to load it's classes instead of the current gMInION_Instance. Thus the dynamic class loading technique is a way of providing extensibility without sacrificing robustness.

gMInION_client

To use the developed web service, it would be enough to distribute the client stubs and let the client figure out how it should be used.

The gMInION_client provides a sample client implementation. It was developed to show how some of the available operations could be combined and it even introduces some extra functionality. The operations described in 3.4.1 which weren't involved in any modifications won't be explained again. These only received a method, which hides the web service call.

In case of the setEnvironment the use of the StoreValues vector was replaced with a Map. The StoreValues vector is only used to make the serialization/deserialization of the environmental variables easier.[5] A Map is considered easier to use.

The runScript method was combined with the inputFileTransfer method, eliminating the possibility that a client wants to run a script, but forgets to transfer this script in the input sandbox.

The gMInION_client provides two methods used to obtain intermediate results of the running jobs. These methods are start_job_debug and runScript_debug. More about how this is achieved can be read in section 3.4.3. Setting the file in the input and output sandbox was also modified. The inputFileTransfer and outputFileTransfer methods set the files which need to be transferred, but because the job is run from the server, these file are staged in- and/or out from/to this machine, not the client machine. Thus the client has to use the saveFileToServer and getFile methods to transfer the input files to the server, respectively to obtain the outputs. These methods were combined in this client module. Even more, when the client sees that the job finished, it tries to automatically obtain all the output files.

To make the system flexible, two ways are provided for transferring the input and output files between the client and the web service. Smaller files are transferred using web service calls, while for the larger files a secure transfer method is provided to/from a storage element. Transferring larger files directly to storage elements lessens the burden

[5]Unfortunately it wasn't possible to use directly a Map, because some inconsistencies were noticed while serialization/deserialization of a Map object while using Axis 1.2 and Axis 1.4

on the server and on the network traffic. Transferring a frequently used file, which doesn't change, to a storage element would make the file reusable multiple time, without transferring it every time. The job description file is automatically adjusted to the transfer method. There is also a configurable limit, which is taken into account when deciding which files should be automatically transferred to a storage element. Although this is a useful characteristic, it makes the client fatter, because more libraries are used, thus these need to be included.

Some recovery mechanisms are implemented, which would allow a client to close the application or even move from one computer to another, but still be able to access the already submitted jobs. Whenever a job is submitted, essential information is stored in an XML file, located at $HOME/.gMinion/gMinionSavedJobs.xml. This file can be read upon restart and the job's output can be retrieved. The following information can be found in this XML file:

- gMInION_Instance identifier
- job identifier
- where the standard error and output files are redirected
- was the job running in debugging mode, if yes, which were the ports where it was listening to intermediate results
- the local job directory where the obtained files are located, after the job finishes

QueryBDII

This is a stand-alone application which can be used by the client, if needed, to obtain information about the available resources. It interrogates the information system (BDII). This module performs a simple query to obtain a list of computing elements or storage elements or a list with computing elements and storage elements which are close to each other. The query is done based on the VO name. The query which can obtain computing elements and storage elements which are close can be useful when we have jobs which need to access a lot of data.

The list with the obtained computing elements can be sorted ascending or descending taking into account three attributes. These attributes can be specified by the user. One of the disadvantages is that these attributes need to be specified as GLUE schema attributes, thus the user has to be familiar with the GLUE schema.

This module was developed only to prove that it is possible to obtain necessary information about the available resources. It should be improved to take into account the job requirements and choose the attributes, which need to be taken into account when matching the job requirements automatically with the resource availability.

After obtaining the list of available computing elements, the client can choose the one that fits best his/her requirements. He/She can then influence the job submission, specifying the server name in the createJob method in the gMInION_client module.

Module interaction

Having given an overview of the main components, we now describe the way they interact with each other. This is shown in figure 3.13.

Before the client interacts with the web service component, he/she should first delegate his/her credentials to a MyProxy service. If he/she wants to obtain information about the available resources he/she should use the QueryBDII application. When the web service component interacts with a client or a workflow engine, it receives SOAP invocations according to the methods described in 3.4.1. Each web service call is processed

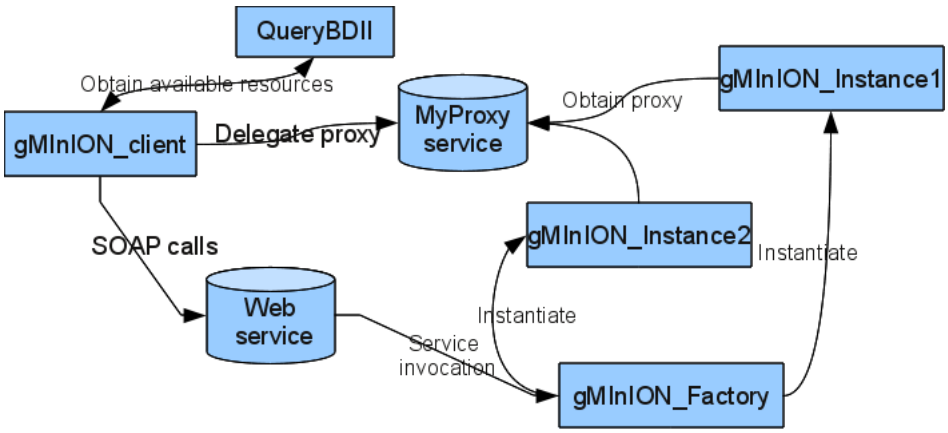


Figure 3.13: Interaction between the main components of gMiNION. The client has to delegate his/her credentials to the MyProxy service. He/She can obtain a information about the available resources and then influence the submission process. To create and submit a job, he/she has to contact the gMInION_Factory usign the gMInION_client. Internally the gMInION_Factory creates a gMInION_Instance for the client. The gMInION_Factory also contacts the MyProxy server to obtain the user's credentials.

in the gMInION_Factory module, which dynamically loads a gMInION_Instance for each client. The gMInION_Instance obtains the delegated credentials from the MyProxy service and manages the user's security contexts. It is also responsible of creating the job description and managing the job life-cycle.

3.4.3 Obtaining intermediate results

In cases, when the user wants to get the intermediate results from a job, the system creates secured communication between the worker node, where the job is actually executed and the client machine. As mentioned in section 3.2.3.2 this is done using openssl s_client and openssl s_server. To obtain these results the client has to make sure that he/she has a valid proxy certificate.

Figure 3.14 shows the job submission using the WMS and the new connection created to obtain the intermediate results.

As we can see the client interacts with the gMInION_client and specifies that he/she wants to obtain intermediate results while running a job. The gMInION_client module performs the following steps automatically when the start_job_debug or runScript_debug methods are called:

1. makes sure that the standard error and standard output files are redirected. In case these weren't previously redirected by the user, their value is set to *stderr<jobID>* and *stdout<jobID>*, where *jobID* is the job's unique identifier.

2. obtain the client's IP address (IP_client) and find two port which can be opened (let's name these port_stdout and port_stderr)

3. a new script is created which contains the executable (script or command), and redirects the standard output and standard error to the client using openssl s_client

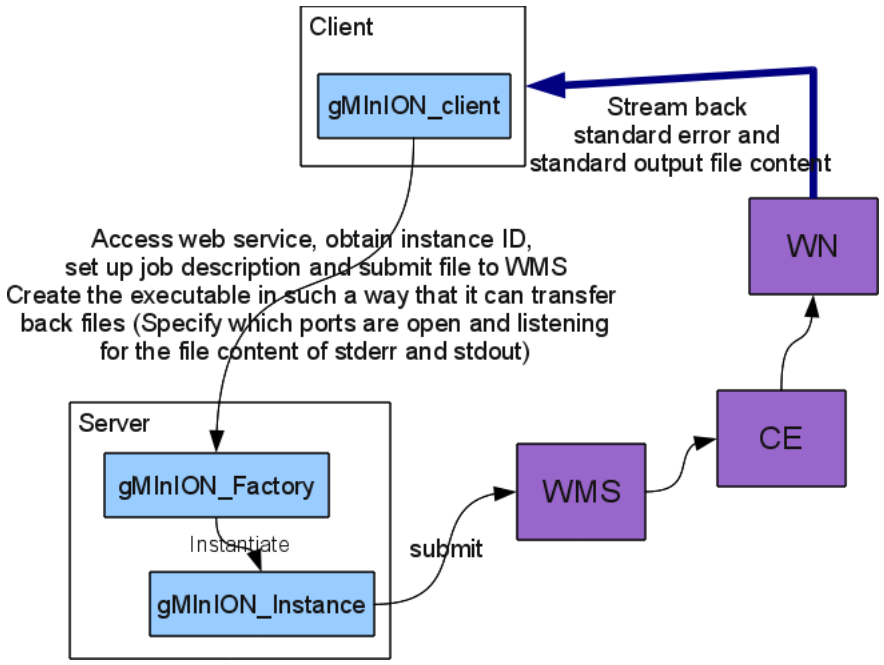


Figure 3.14: Obtaining intermediate results. The job description specifies the ports where the client is listening for the intermediate result. After the job is submitted, the executing host has to connect to the client and stream the required files.

commands. To ensure that the openssl commands execute successfully the directory where the CA certificates are stored needs to be known on the worker node. This isn't a big drawback, because the CA certificates standard location is the /etc/grid-security/certificates directory. The IP address of the client and the previously established ports are also written in the new executable script. For example such a script looks like:

```
((/bin/bash <executable_script> >&1 ) | tee stdout<jobID> |
 openssl s_client -connect IP_client:port_stderr -CApath
 \$X509\_CA\_CERT 1>/dev/null 2>/dev/null) 3>&1 1>&2 2>&3 |
 tee stderr<jobID> | openssl s_client -connect
 IP_client:port_stdout -CApath \$X509\_CA\_CERT 1>/dev/null
 2>/dev/null
```

In case the job wants to execute a single command and not a script, the command is specified instead of /bin/bash <executable_script>

4. use the runScript command to transfer and start running the job

5. using two new threads the client starts to listen on the *port_stdout* and *port_stderr* using openssl s_server. The output of the openssl s_server command should be transferred to the stdout<jobID>_dbg and stderr<jobID>_dbg files. This is done by executing the following bash command on the client machine:

```
openssl s_server -accept <portNr> -CApath
 /etc/grid-security/certificates  -key <proxyFile> -CAfile
```

<proxyFile> -cert <proxyFile> -quiet >> <stdout<jobId>_dbg OR stderr<jobId>_dbg>

This command is started from the gMInION_client module using Java's Runtime class, which allows the application to interface with the environment in which the application is running. The gMInION_client module is responsible for specifying the correct port number, the location of the user's proxy certificate and the directory where the CA certificates can be found.

As already mentioned, in case the client crashes, after restarting it can read the $HOME/.gMinion/gMinionSavedJobs.xml file, and see if a job is still running. In case a given job is running, it can be seen if it was started in debug mode or not. If intermediate results were requested, the client can start listening on the same ports, thus being able to receive the message which are sent by the worker node. Unfortunately the messages sent while the client wasn't online will be lost. These results will be obtained when the job finishes and the final results are received. This functionality can be further extended as future work.

Chapter 4

Experimental setup and discussion

The main objective of this work was to create a middleware agnostic layer to access the underlying computing resources. This generic approach is useful for users, providing them a unified way to access multiple middleware systems. However what's gained in flexibility may be lost in performance. It is likely that this general approach will introduce some overhead when jobs are managed using the gMInION module.

The target of this performance analysis was to determine the overhead introduced by the gMInION module. Tests were performed by submitting jobs to the LCG-CE, CREAM CE and WMS. The performance of the file transfer mechanism was also tested. The goal was to see if transmitting larger files directly to a storage element and not through the web server would result in a quicker transfer. The overhead introduced by using a secured web service was also measured.

The setup for the experiments was as follows: the web server, in which the gMInION web service is running, was set up on a virtual machine on a Dell 1950 with 4 cores. The VM had 3 GB memory and access to 3 processors.

For running the test programs the production systems at Nikhef and SARA were used. This means that other users were using the same systems, thus competing for the same resources. Running the tests on production systems provided us with realistic environment and response times.

4.1 Comparing direct submission with submission using the gMInION abstract layer

A performance analysis has been performed by submitting and executing jobs. Comparisons were made between executing jobs directly, by contacting the scheduling elements and by using the gMInION abstract submission layer. Jobs can be submitted to the WMS, to the CREAM CE and to the LCG-CE. For all the jobs running on a specified scheduling element type, the same scheduling element was used. This decision was made to limit the network latency difference which could occur in case the gMInION module would choose a WMS at a different location. In case of the direct job submission, the client has to know exactly where he/she wants to submit the job.

All test programs were written in Java, and the same APIs were used both for direct submission and for job submission using gMInION. This allows us to measure the over-

head of the abstract layer introduces, and not the overhead introduced by other modules. In all cases the users credentials were obtained from the same MyProxy server.

A parameter sweep job was mimicked, which after submission, sleeps for 50 seconds. This job was submitted in all the tests, thus all the jobs have similar characteristics. The job's attributes have to be specified either in a middleware specific job submission file (for direct submission) or the client has to invoke the necessary methods (in case of gMInION), thus setting up the job description by performing several SOAP calls to the web server.

Both submission and execution time was monitored. In case of the execution time the queuing time[1] has to be taken into account, which can severely influence the execution time. Unfortunately the load of the scheduling elements couldn't be controlled, thus the execution time results can vary quite a bit. The execution time includes the submission time, the queuing time, the actual execution and the time required to obtain the output files.

Submitting to the WMS

From figure 4.1(a) it can be seen that the submition time with gMInION appears to be smaller. This is due to the implementation of the submit method. This method doesn't wait until the scheduling element replies to the request. It is an optimistic way of submitting jobs, because it presumes that the job description was set up correctly and that the scheduling element will be able to execute the job. This is realized by creating a new thread for every job. This thread is responsible of starting the job on the scheduling element and monitoring the status of the job.

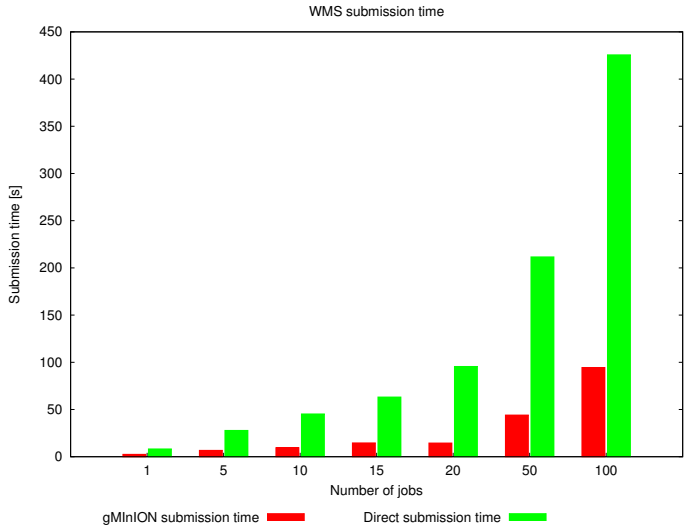
There is a configurable limit which states how many jobs a user can run concurrently. This results in a different case: when the user tries to submit more jobs than this limit, the extra jobs are queued, and they wait until they can start executing. Although these jobs aren't sent to the scheduling element, they are only in the submission system, a response is sent to the client. This way the client can continue his/her work by creating and submitting new jobs. The advantage of this optimistic approach is that the client doesn't have to wait too long for the job to start. From the moment the client starts the job, the gMInION module is responsible of managing this job.

On the other hand this optimistic approach can cause extra overhead. This can happen when the job description is incorrect, for example, if it is missing some essential attributes. In this case the gMInION module responds to the client stating that it will try to submit the job. The scheduling element eventually will discover that the job can't be executed and will propagate this error back to the gMInION module, which should propagate it back to the client. Unfortunately, the client won't receive this feedback as soon as the scheduling element discovers the error, because the gMInION module only interrogates the state of the job periodically. In case the scheduling element changes the state of the job right after the gMInION client interrogated this state, the state will be change in the gMInION module only after waiting some time. Even after the gMInION module has the correct state of the job, it still can't push it back to the client. It has to wait until the client interrogates the state of the job. The same behavior can be expected if the system can't find a suitable resource to run the job on. Although this is not very likely, because we're addressing a huge number of underlying resources by using the middleware agnostic layer. Another case when the job won't be able to execute is when the client's certificate expires while the job is in the queue of the scheduling resource, so

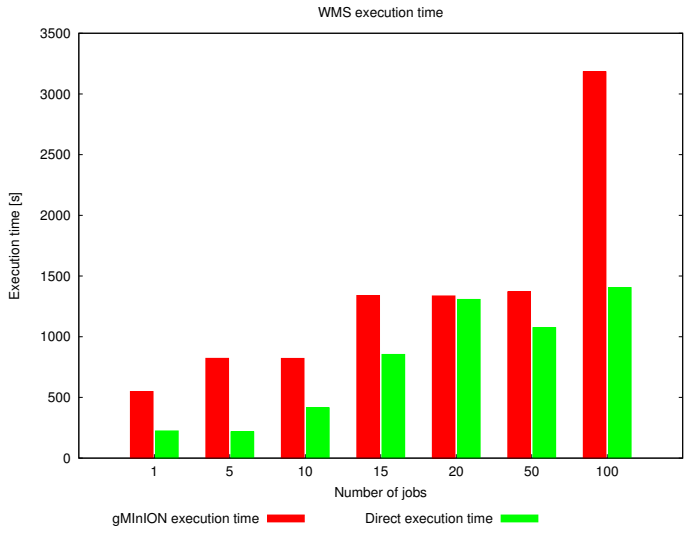
[1] At the WMS or CE, the job is kept in a local job queue, until it is submitted to a specific worker node, where it will be executed. Thus the queuing time depends on the background load.

before the job execution is started on the executing system. This would also introduce the discussed overhead.

In case of direct submission to the WMS the submission time is higher, because the client has to wait until the WMS schedules the job. This means that the client has the opportunity to resubmit the job as soon as the WMS fails to execute it. However in this case the client is responsible of implementing a fail-over mechanism, while with the gMInION module this can be implemented once and hidden from the clients.



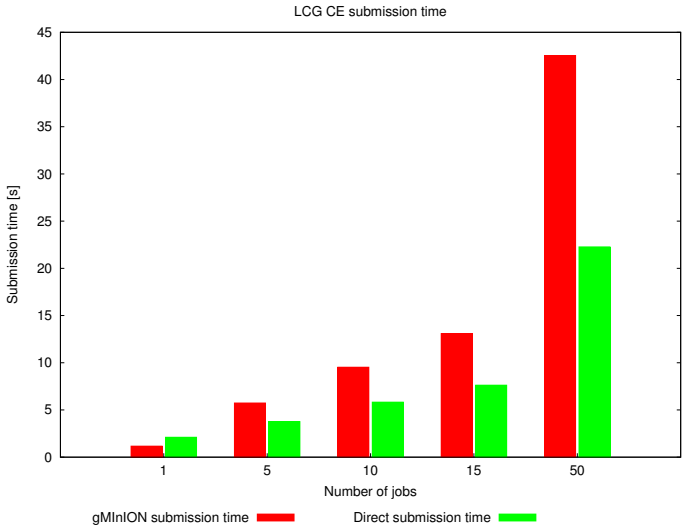
(a) Submission time



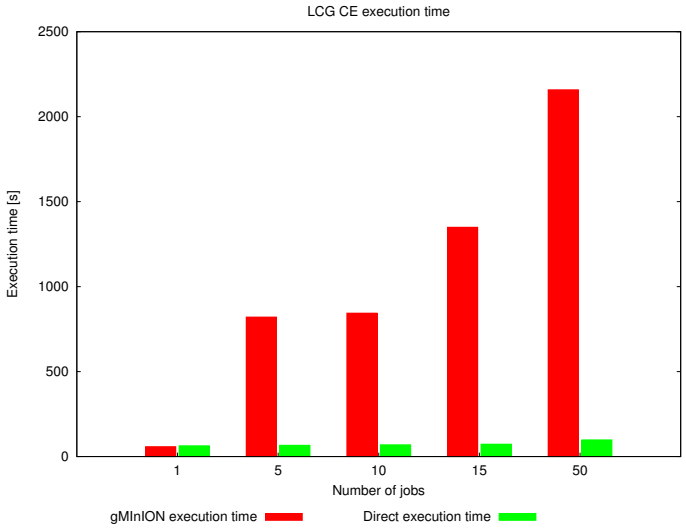
(b) Execution time

Figure 4.1: WMS - direct job submission vs. jobs submitted with gMInION

In figure 4.1 we can observe how the submission and execution time is affected by specifying how many jobs can be run concurrently by one user. In the described test



(a) Submission time



(b) Execution time

Figure 4.2: LCG CE - direct submission vs submitting jobs using gMInION

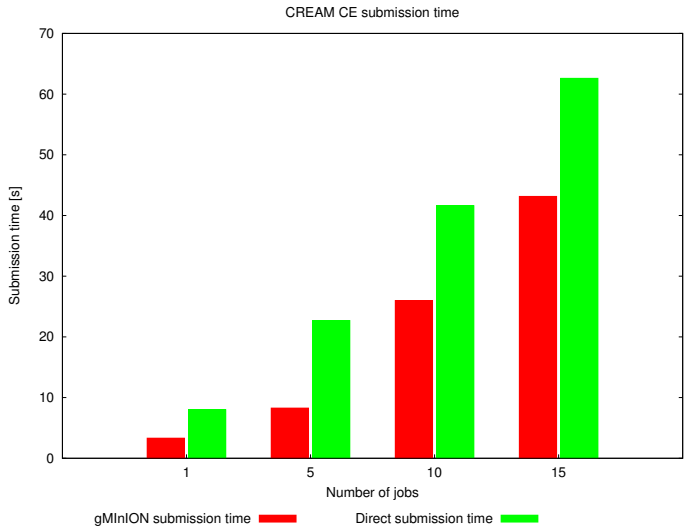
In figure 4.2 we can see that the submission and the execution time when submitting jobs with gMInION is higher than the submission and execution time when submitting jobs directly to an LCG-CE. This is caused by the fact that the gMInION module is submitting jobs to the LCG-CE through the WMS. Even if the queue is known when the jobs is submitted, the scheduling has to be done through the WMS otherwise the results cannot be transferred back, only by copying them one by one using a specific protocol. If this approach would have been taken, our module wouldn't be middleware agnostic. We did not dedicate much time to solve this problem, because LCG-CE is going to be replaced by CREAM CE.

If we compare the direct execution times between the LCG-CE and the WMS runs, we see that the LCG-CE performs better. This is due to the fact that the waiting queue

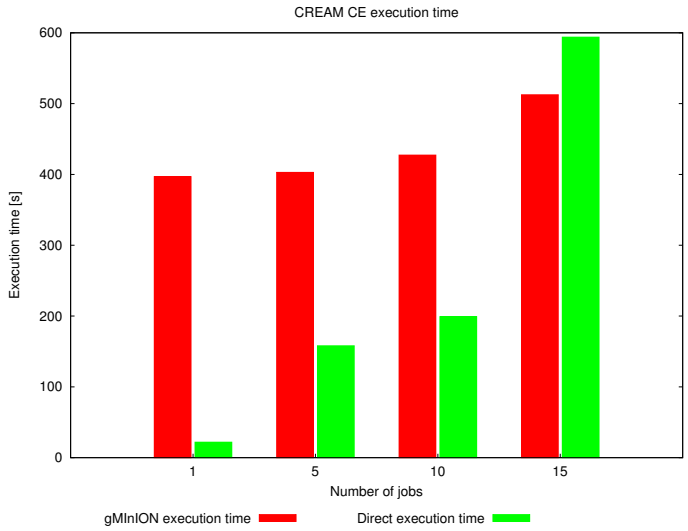
at the WMS is skipped.

Submitting to the CREAM CE

In these tests we bypass the high level scheduler (WMS). This means that we have better interaction with the cluster and the submission and execution times should be faster. However, as mentioned in the case of the LCG-CE tests, the user has to know which scheduling element is available and can provide the requested resources necessary for the job that's going to be submitted. We expect the CREAM CE to perform better than the LCG-CE, since one of it's purposes is to replace the LCG-CE in the near future.



(a) Submission time



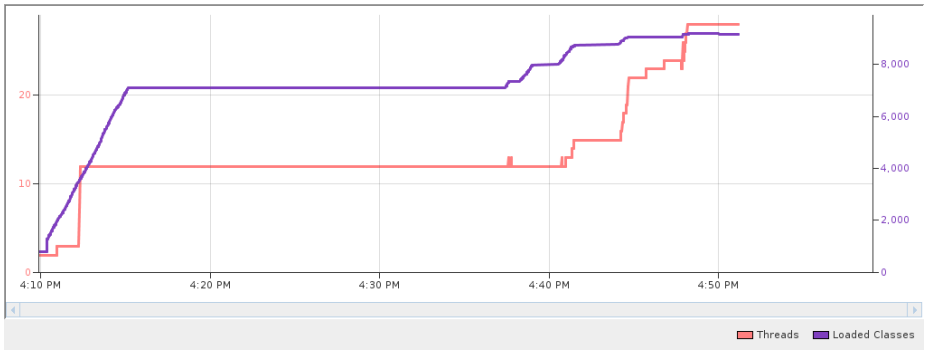
(b) Execution time

Figure 4.3: CREAM CE - direct submission vs. submitting jobs using gMInION

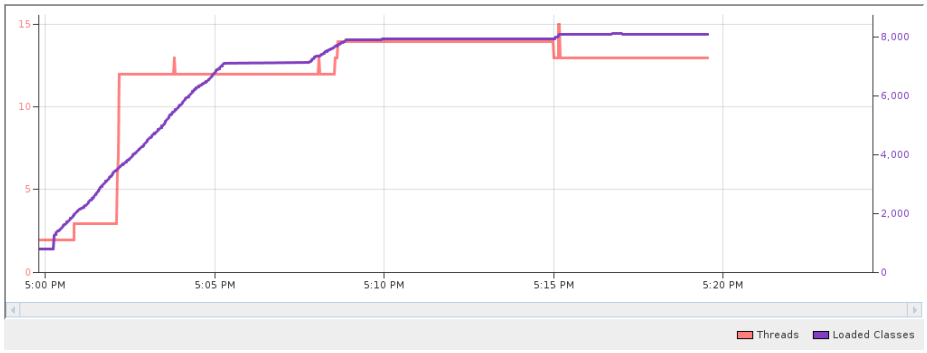
In figure 4.3 the submission time using gMInION is shorter than the direct submission time. This is caused by the same optimistic mechanism explained in case of the WMS submission. The execution time is longer, because of the waiting time between two interrogations of the job status.

Figure 4.3(b) shows the execution time. The seen differences between execution with gMInION and submitting a job directly to a computing element are produced by the queuing time on the scheduling element. The execution time in case of gMInION is higher because of the time a job has to wait after it finished on the scheduling element, but it's result and status weren't retrieved by the gMInION module and by the client.

Unfortunately with the CREAM CE API only a restricted number of jobs could be run. This was caused by the used CREAM CE API. If more jobs were started than 15, because or the Java heap ran out of memory or the garbage collection couldn't be perfomed, which also lead to out of memory error.[2] It is surprising that the Java JVM ran out of memory, even though the maximum allowed head size was 2GB. To see why this behavior occurred, some extra checks were made to find out what is causing the system to fail. The whole project was profiled using the Netbeans profiler [2] and an increase in memory was seen because of the CREAM CE API. To see the difference and discover the cause of this error the memory consumption and the running thread numbers were compared between running the same job on CREAM CE and on WMS, while the profiler was on.



(a) CREAM CE



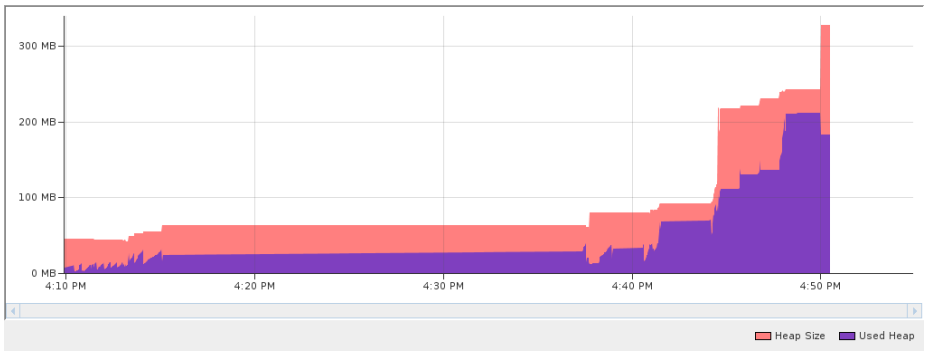
(b) WMS memory usage

Figure 4.4: Number of threads for one job

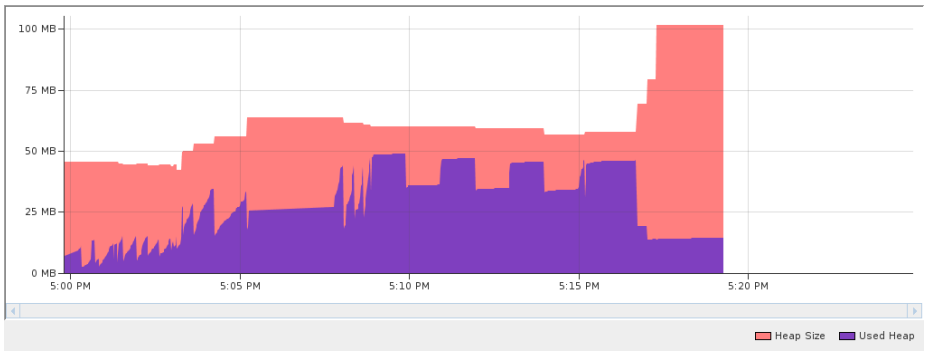
[2]The following errors were encountered: "java.lang.OutOfMemoryError: Java heap space" or "java.lang.OutOfMemoryError: GC overhead limit exceeded"

Figure 4.4 presents the number of active threads after a job finishes. These include the threads which are required for Tomcat to run and the threads used by the gMInION module. Tomcat and gMInION start a (more or less) fixed number of threads. Thus the difference between the number of threads shown in figure 4.4(a) and 4.4(b), which is around 12 threads is probably because of the CREAM CE API.

Figure 4.5 shows the memory consumption. When running with the WMS Java API, after the job is run, the memory usage decreases, without us invoking the garbage collection. This isn't the case with CREAM CE. The last decrease in figure 4.5(a) is because we forcibly invoked the garbage collection. Both graphs include the memory usage of tomcat and of gMInION.



(a) CREAM CE memory usage



(b) WMS memory usage

Figure 4.5: Memory usage for one job

The seen difference is about 150 MB. A rough calculation shows us that this would require about 2 GB of memory if 15 jobs would be ran concurrently. From these graphs it was concluded that the CREAM CE API isn't releasing all the used resources and it produces a memory leak. Hopefully there will be solution for this problem soon and then more jobs can be submitted to the CREAM CE resources.

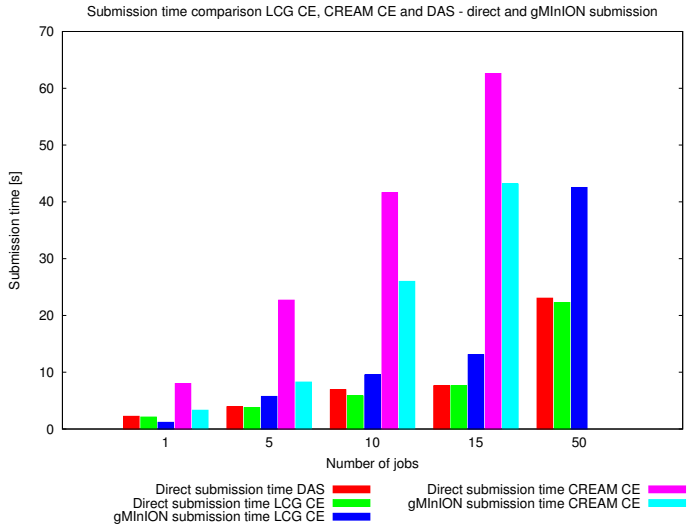
The same behavior was seen when running jobs without the gMInION module.

Compare submission to DAS, LCG CE and CREAM CE

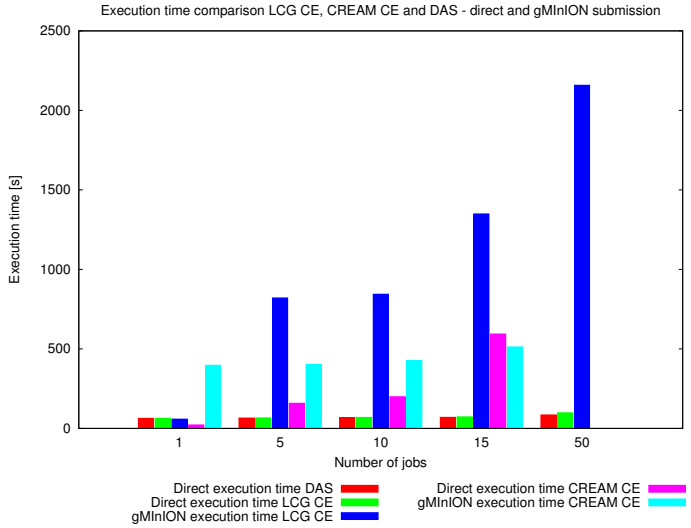
This section compares the previously presented results to direct submission to a cluster, namely Distributed ASCI Supercomputer 3 (DAS3) [4]. The Distributed ASCI Supercomputer 3 is a Dutch testbed that links together 5 clusters at 5 different sites. Its goal is

to provide infrastructure for research in distributed and Grid computing. Programs are started on the DAS3 compute nodes using the Sun Grid Engine (SGE) batch queueing system. The SGE system reserves the requested number of nodes for the duration of a program run. It is also possible to reserve a number of hosts in advance, terminate running jobs, or query the status of current jobs.

These test were performed to show the delays which can be encountered when submitting to computing elements and not to cluster queuing systems. This can be useful for the clients using cluster systems to see the overhead introduced by using a Grid environment.



(a) Submission time



(b) Execution time

Figure 4.6: Comparing direct job submission with submitting jobs using gMInION to the LCG CE, CREAM CE and DAS

As expected, the submission time to a cluster is the fastest, as seen in figure 4.6(a).

Although the direct submission time with to the LCG-CE is quite close. This speed can't be achieved by submitting with the gMInION module, because gMInION can only submit to the LCG-CE through the WMS. It is surprising that the direct submission to the CREAM CE takes a lot longer. Unfortunately we couldn't submit 50 jobs to the CREAM CE because of the reasons explained in the previous section. We can notice in these graphs too, that the execution time with gMInION is longer, probably because of the waiting time between two interrogations of the status. Part of this time difference can be explained by the delay caused by the WMS and the background load.

4.2 Submitting jobs to multiple middleware systems

These tests were performed on a different web server. This machine has 8 Intel(R) Xeon(R) CPU 3.00GHz and 32 GB of memory. Tomcat was allowed to access 10 GB of memory.

The test setup was as follows: a list of available computing elements and WMSs were obtained using the users VO information. The submitted job consisted of an executable which required the worker node to sleep for 50 seconds. After specifying the number of jobs which had to be run, the computing element was chosen randomly from the previously obtained list. In case the computing element was an LCG-CE, a WMS was chosen randomly. Then the job was submitted.

In the first run 91 jobs were submitted. Time was measured after the submission and execution of 1, 5, 10, 25 and 50 jobs. Unfortunately only the first 4 time measurements were performed successfully. Before reaching the last measurement
"java.lang.OutOfMemoryError: Java heap space" was received. From the first 41 jobs, 33 were submitted to the CREAM CE and the remaining 11 to LCG-CE.

Because of the memory leak in the CREAM CE API, around 4.8 GB was already occupied. Choosing a CREAM CE as the scheduling resource had a higher probability, because at the moment more CREAM CEs are in production than LCG-CEs. So it is highly probable that other 40 jobs were tried to be run on CREAM CE which resulted in memory overflow.

The second run could be done after the tomcat server was restarted. This run consisted of 50 jobs. The measurements were only taken after all the jobs finished. From these jobs, 32 ran on CREAM CE and the remaining 18 on LCG-CE. The submission and execution time of these runs can be seen in figure 4.7.

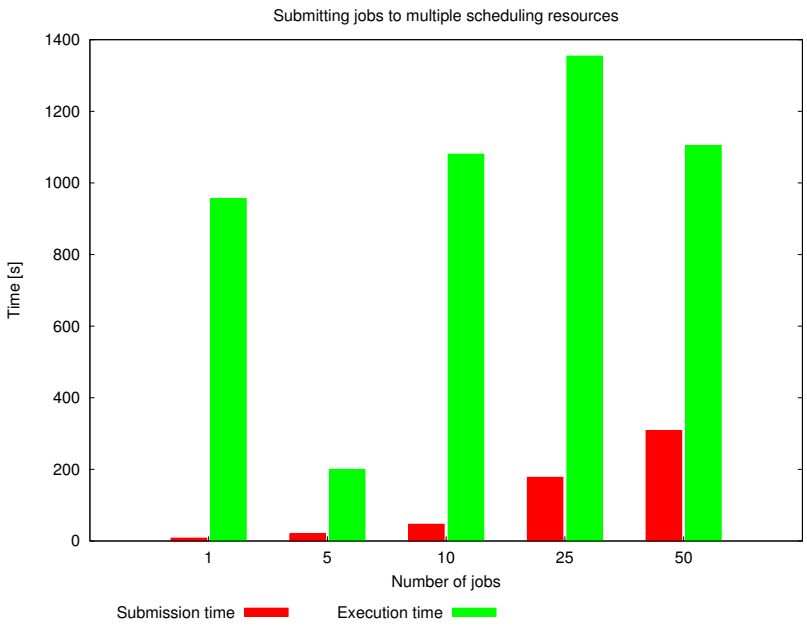


Figure 4.7: Running jobs on multiple scheduling systems

In figure 4.7 the difference between the execution time of the first job and the following 5 jobs can be explained not only by referring to the background overload. The first job was run using the LCG-CE resource scheduler, thus it had to contact the WMS and wait in its queue, while the next 5 jobs were run on the CREAM CE, which was contacted directly by gMInION.

The execution time of the following 10, respectively 25 jobs is probably due mostly to the background load, because only a limited number of jobs were scheduled to the LCG-CE.

4.3 File transfer

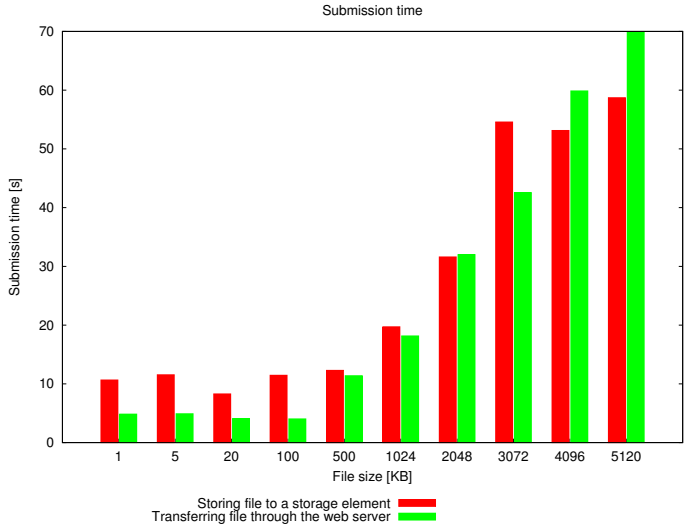
Due to potentially long input file transfer times, the actual start of a job's execution may be much later than the time when the processors was allocated for the job. This means that much processor time is wasted.

gMInION offers two ways of transferring files. Smaller file can be transferred through the web server, in which case the client can be thiner. In this case no extra libraries are needed which can contact the storage elements. This is useful when a user wants to transfer small files which change often.

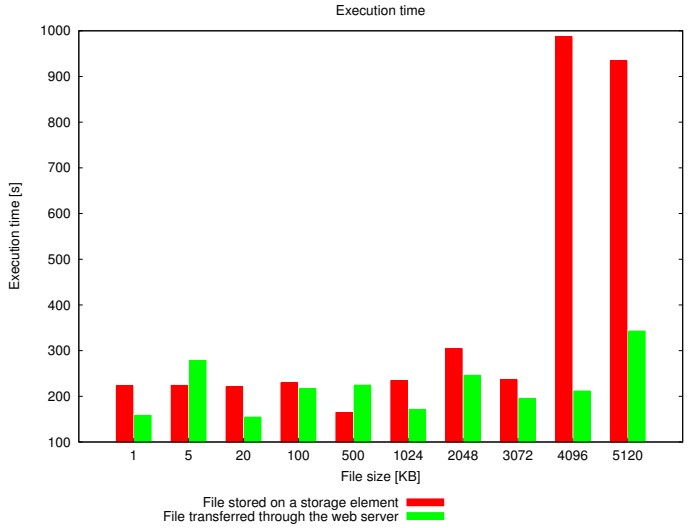
If large files are transferred, it is assumed that transferring these directly to a storage element would minimize the network load and allow the jobs easier and quicker access to these files. The drawback of this approach is that a secure connection has to be established with the storage element too (not only with the gMInION service). This could mean a lot of overhead if the files are small, not reused and change often. Transferring files directly to a storage element would also be beneficial, if the files can be reused, because their content doesn't change.

Tests were performed with different file sizes. And the same jobs were run with the file transferred to a storage element and with the files transferred together with the job

to the web server.



(a) Submission time



(b) Execution time

Figure 4.8: File transfer directly to the storage element or through the web server

It can be noted that when files are transferred through the web server, all the files need to be transferred several times through the network. First a transfer is made from the client to the web server, then the web server stages in the files to the scheduling element, which then transfers the files to the worker node. If the client transfers the files directly to the storage element, the job description can be written in such a way that the worker node could obtain the files directly, without these being transferred to the scheduling element.

In figure 4.8 the time necessary for the file transfer to be executed is included in the

submission time. As seen in figure 4.8(a), the submission time for smaller files through the web service is shorter than the submission time through a storage element. This can be caused by the extra authentication step required by the storage element. For larger file the situation changes, because the methods used to transfer the files to the storage elements are specialized for high throughput transfer. We still have to take into account the response time of the storage element, which can introduce extra overhead. This can be seen when transferring a 3MB file.

The execution time shown in 4.8(b) includes the queuing time, the time necessary to transfer the data from the storage element to the worker node and the execution time. On figure 4.8(b) we can notice an increase in the execution time for files larger than 4MB. This can be caused by the fact that the data transfer between the storage element and the system executing the job takes up a lot of time. This can't be seen if the file was transferred through the web service, because in this case the file is located in the input sandbox of the job (so it is considered as an input file). This means that the job is transferred to the job, before the job begins.

Although choosing to transfer all the file through the web service may seem like a good solution, however this can't be used all the time. Sometimes users don't have all the files on their local machines. They may need to access too many files or the files may just be too large. In these cases the job description can be created by taking into account that the input files are on a storage element. This means that these files don't have to be transferred to the web service, so the submission time would decrease.

Transferring many files through the web service has a considerable disadvantage. In this case we are using the machine which hosts the web service as a storage element. If too many files are transferred the storage limits of this can be easily reached.

We only discussed input file transfer, but the same ideas are valid for output transfer too. The job description can be specified in such a way that the results are directly transferred to a storage element or the gMInION web service can obtain the output files from the scheduling element. If the results are transferred to the storage element, the client can access these using specialized protocols, after he sees that the job is done. If the files are transferred to gMInION, the user has to request the results from the web service after the job is done.

For some jobs, if we want to obtain the output several times it could be beneficial to transfer the output to a storage element. This way the scheduling element would be able to finish the job quicker, without keeping the output files, so it would be able to release some resources. The user or group of users would be able to access these files later, without rerunning the tests if they lose their local copy of the results. And we didn't even mention the fact that files on a storage element are managed by highly skilled professionals, who are responsible for backing up these files.

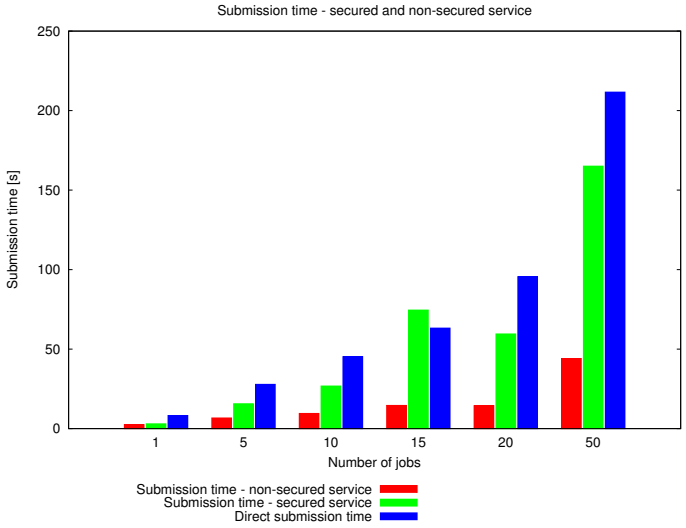
4.4 Comparing the performance of a secure and a non-secure web server

The following test was performed to show the difference between submitting to a secured and a non-secured web service. This test is considered interesting especially for workflow manager systems. In this case the web server could be setup on the same machine as the workflow manager and the access to it could be restricted to localhost. This way no other security measures need to be taken.

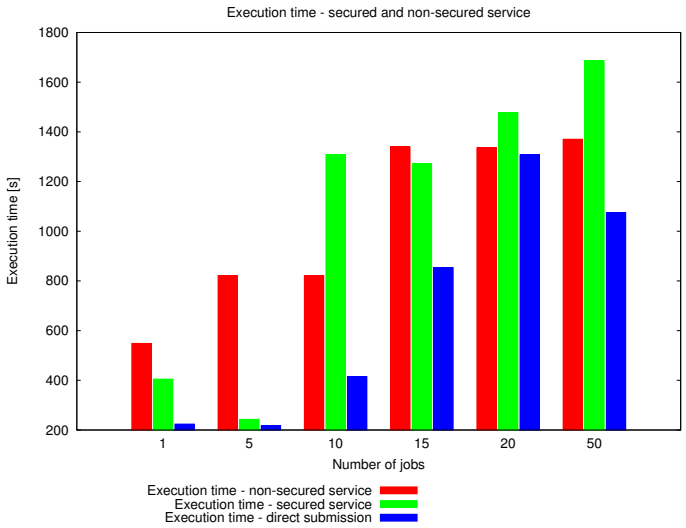
Figure 4.9 shows the added overhead by creating a secure channel between the server and the client. The mutual authentication was achieved by Tomcat with SSL and using the available Grid certificates on the server and on the client side.

Especially the submission time is important, because there's where we can see the connection establishment. Although the secured service adds some overhead, this is still acceptable and compared to direct submission to the WMS it still performs better due to its optimistic scheduling.

The differences in the execution time are due to the background load.



(a) Submission time



(b) Execution time

Figure 4.9: Secure and non-secure web service tests - submitting to WMS

Using Wireshark, the network traffic was measured. We can clearly see the overhead introduced by the secure connection in figure 4.10. We can also observe the number of SOAP requests sent to the web server to set up the job description and to start the job. In figure 4.10 the green line represents the traffic sent from the client to the server, while

the red line shows up the traffic sent by the server to the client.

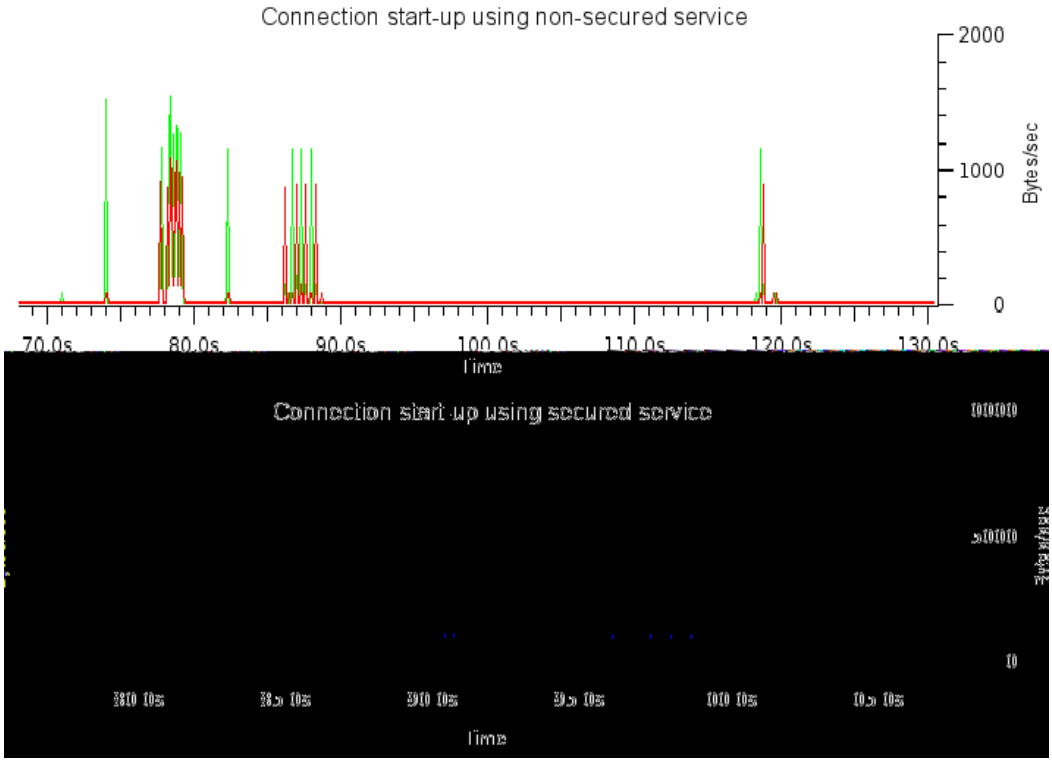


Figure 4.10: Network traffic for secured and non-secured service

It can be seen that in case of the secured service, the server initializes a mutual authentication session, which involve some traffic, and afterwards the job description is set up and the job is started.

In case of the non-secured service, the last traffic that we see (at 120s) is already requesting the status of the job.

4.5 Number of clients

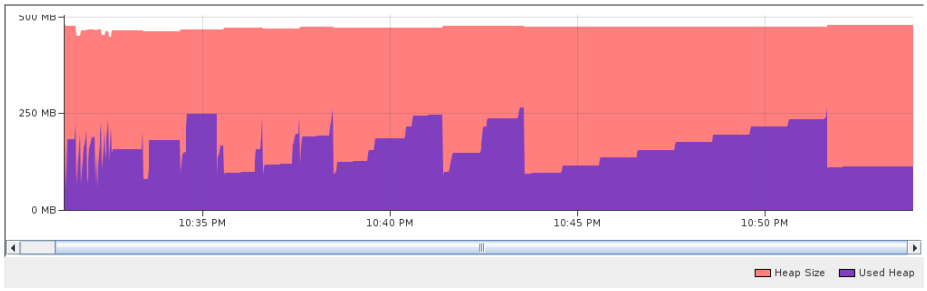
Another interesting measurement is to see how many clients can connect to the web server at the same time. The setup was the following: The VM was used and the tomcat server could access 2GB of memory and the PermGen space was set to 1GB. The PermGen space had to be set higher, otherwise with 120 connections we received OutOfMemory

exception. [3]

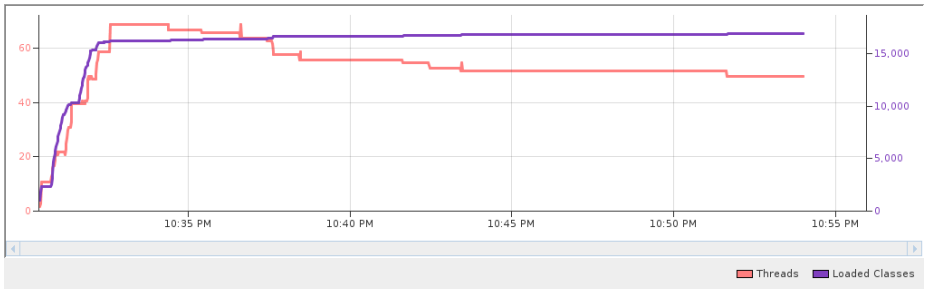
First 50 clients were started with a 10 second interval between each consecutive client. When we saw that these jobs were started, this process was repeated twice. Each job was submitted to the WMS and had to sleep 10 minutes on the assigned worker node, thus making sure that the jobs won't finish before we start the second batch of 50 clients.

To show the memory usage and the number of created threads the test was repeated with fewer clients, while the profiler was running. First 10 clients were started and after they started executing another batch of 10 clients was started. These results can be seen in figure 4.11.

In figure 4.11(b) we can clearly see when the second batch jobs was started and when some of the clients finished. On the same figure we can see the number of loaded classes too. On figure 4.11(a) the peaks are because of the jobs which are starting and the jobs which are interrogating their status. The jobs which were running when the profiler was active, only waited for 2 minutes.



(a) Memory usage



(b) Thread number

Figure 4.11: Number of clients

It can be seen that even more clients could connect to this server. Unfortunately if

[3]The Java Heap is organized in generations for optimized garbage collection. Generations are memory segments holding objects of different ages. Garbage collection algorithms in each generation are different. Objects are allocated in a generation for younger objects - the Young Generation, and because of infant mortality most objects die there. When the young generation fills up it causes a Minor Collection. Assuming high infant mortality, minor collections are garbage-collected frequently. Some surviving objects are moved to a Tenured Generation. When the Tenured Generation needs to be collected there is a Major Collection that is often much slower because it involves all live objects. Each generation contains variables of different length of life and different GC policies are applied to them. There is a third generation too, the Permanent Generation. The permanent generation is special because it holds meta-data describing user classes. Applications with large code-base can quickly fill up this segment of the heap which will cause java.lang.OutOfMemoryError: PermGen no matter how high your -Xmx and how much memory you have on the machine. [55]

more jobs were started, the profiler slowed down the process and after a while crashed the server. At that moment no error messages could be seen. As already mentioned, without the profiler 150 clients were run concurrently.

4.6 Integrating with WS-VLAM

The current WS-VLAM [45] architecture is composed mainly of two parts, the WS-VLAM composer implemented as a client application and a set of WSRF services deployed in GT4 Container. These include standard GT4 services such as the Delegation Service, and a set of WSRF services developed in the Virtual Laboratory for eSciene project namely; the Workflow Components Repository (WCR), the Resource Manager and the Runtime System Manager (RTSM).

To solve the authorization and authentication problems, the GT4 Delegation Service is used to delegate the necessary credentials for job submission to the resources. The execution engine is wrapped as a Run Time System Manager (RTSM) Service. This consists of two parts: RTSM Factory Service (RTSMFactory) and RTSM Instance Service (RTSMInstance). The factory is a persistent service which instantiates a transient RTSM Instance service whenever a user submits a workflow execution topology.

The client is also capable of monitoring and controlling the execution of a running workflow execution using a unique identifier () for the RTSM instance, obtained from the RTSM Factory.

The job submission in the current architecture is done using the GT4 resource manager (GRAM). The description of each module and the list of callbacks are obtained and then a task description is created, which is placed in a queue with a fixed length. If there is no space left in the queue the method blocks. After receiving the confirmation from GRAM, stating that the job has been actually submitted, the task description is removed from the local queue. A Global Access to Secondary Storage (GASS) server is used to redirect the standard output and error streams of a module.

Workflow tasks rely on resource co-allocation since tasks must synchronize on communication channels before actual execution starts. The communication component provides a communication channel between a workflow component and the RTSM. To avoid inbound restrictions to the worker nodes, modules initialize the connection to the RTSM.

gMInION is used to replace the existing submission module. In the following paragraphs some of the encountered integration issues are presented.

Adding VOMS attributes
Currently WS-VLAM uses GSSCredentials to submit jobs to GT4. To be able to submit to other type of middleware too, VOMS attributes have to be inserted into the certificates. In this section code snippets can be found with which the VOMS server can be contacted and a VOMS certificate can be obtained. The vomsjapi has been used to implement this method. The listing of this method can be seen in Appendix B.

File transfer
As mentioned in section 3.4.2 the implemented gMInION_client is capable of transferring input and output files using two methods. Unfortunately one of these possibilities can't be used by the submission module used in WS-VLAM. It isn't possible to store some files directly to a storage element, because that part of the code requires some libraries which can't be deployed in the GT4 container. More specifically this is caused by the Axis library. The GT4 container uses Axis 1.2, and this version can't be changed within the container. To transfer files Axis 1.4 version is needed.

This leaves us with one way of transferring files between the RTSM client and the Tomcat server. The content of the file will be transferred as a byte array in a SOAP message.

File observer
The GUI of the WS-VLAM offers the possibility to see the stdout and stderr outputs while the job is running. It would have been a pity to lose this is a nice feature, when the submission module is replaced.

This feature was implemented with the help of the Global Access to Secondary Storage server. This component is part of the GT and it simplifies the porting and running of applications that use file I/O to the Globus environment. Using this server the standard output and standard error can be redirected.

With a middleware independent layer this solution had to be altered. As explained in section 3.4.3, it is possible to obtain intermediate result, if the jobs are started in debug mode. Thus all the jobs in WS-VLAM are started using either the start_job_debug or the runScript_debug methods from the gMInION_client module. We only have to make sure that there is a valid proxy certificate on the machine where the RTSM engine is running and that this certificate is used to set up the mutual authentication with OpenSSL. This way the machines where the jobs are executed will send back to the machine on which the RTSM engine is running the stderr and stdout files periodically. This way we can obtain intermediate results, while the jobs are running, without being concerned about the underlying Grid middleware.

The only thing left to do is to bind these files to the GUI. This can be done using the observer pattern. Every time new data is received from a worker node, the GUI has to be notified to update its interface. We can use a thread which constantly checks if the content of a file changed. When it did, then it sends a notification to all the object which subscribed to watch its state, in our case the object which is responsible of displaying the GUI.

At present tests couldn't be performed using WS-VLAM , because the submitted workflow components aren't correctly executing an RPC call to the RTSM engine. The submission of a workflow component is done by sending a script to a worker node, which has to obtain the archives, containing the components' implementation, from a repository. Then it has to execute the obtained code. Within this code the RTSM engine has to be contacted. Tests were done without the RTSM connection and it could be seen that the jobs are scheduled and they obtain the requested archives. It has been concluded that the RTSM modules need to be investigated to discover the exact cause of this error. Due to time limitations this couldn't be further investigated.

Chapter 5

Conclusions

Scientific and engineering experiments involve people and facilities that are distributed across organizations and across the globe. During the last decade there has been a growing interest in Grid computing, which is increasingly being adopted in many scientific projects due to its ability to enable collaboration and access to resources. Unfortunately there are several Grid middleware systems which enable access to the different underlying resources. These middleware systems work in isolation. Users have to build their applications in such a way that it addresses all the middleware systems their intend to access. Running jobs on a given Grid middleware involves addressing problems like: authorization, authentication, file pre- and post-staging, monitoring and data management. Most of the existing Grid systems deal with all these issues, but they provide different ways to obtain the same results. It's only a matter of time until user's wish to access multiple middleware systems, without understanding the middleware specific underlying details. Due to the complexity of the Grid infrastructures and the evolving standards, abstractions are needed to hide the problems originated from the different Grid middleware architectures.

This thesis presented the design and implement of an abstraction layer which solves the problems encountered during a job's life cycle in several existing Grid middleware systems (gLite, Globus) and which can provide a user-friendly and unified way to gain access to all the available resources.

An extensive literature study was performed to evaluate the available APIs which stated to be able of submitting middleware independent jobs. JSAGA was the only system which performed the job and credential management operations in an acceptable way addressing both the gLite and the Globus middleware systems. It's main drawback was that it is meant for single user environments and it doesn't provide interaction with the running jobs. Because of JSAGA's features, this system was further developed to provide all the required features for advanced job monitoring and it was integrated as a plug-in in the developed application. This way a workflow engine can also benefit from the advantages provided by JSAGA.

The developed system has a service-oriented architecture combined with a plug-in architecture. Using a web service interface clients can create and manage their jobs without being aware of the complex nature of the underlying middleware systems. The resulting application is a stand-alone web service, which can be used to submit independent jobs or jobs orchestrated by a workflow manager system. Intermediate result files can be obtained without knowing on which middleware system the job is executing.

The presented results prove that the introduced overhead within acceptable limits, especially if we take into account that using this module a client application could easily migrate from using one Grid middleware to another, or even using several middleware

systems at the same time.

The abstract job submission module can alleviates one of the shortcomings of workflow manager systems, namely: most of the workflow manager systems can only submit jobs to one underlying Grid middleware system. The abstract layer is only responsible for submitting and managing the jobs and the corresponding files. The workflow orchestration, respectively determining the dependencies between the jobs should be done by other modules of the workflow manager.

The presented system and literature study provides a solid background for further research in the area of middleware independent job submission. It also has a practical contribution, proving that it is not impossible to decouple the client application from the underlying infrastructure. Using gMInION would allow changing the underlying infrastructure, without influencing and involving the clients, which can be workflow engines too.

5.1 Future work

Future work can focus on developing other plug-ins for underlying architectures which weren't addressed, for example cloud computing resources or the Condor or the UNICORE middleware.

A more advanced resource selection module should be developed. While performing the matchmaking task between the job and the available resources, the specified job descriptions have to be taken into account. This would allow selecting a resource which has the required hardware and software specifications.

The resource scheduling module should be invisible for the client. Every time a job is submitted, the resource selector could provide the best resource match for the job. Such a resource manager should be used in case a client wants to transfer files to a storage element. In this case a storage element could be chosen which is closer to the computing element where the job will be executed.

A mechanism for "intelligent" handling of failed jobs could be developed. This would require submitting the failed job to a different machine. The resource scheduling module shouldn't choose the machine where the job failed, even if this is the 'best' match for the job's specifications. Depending on the cause of the failure it could also choose a different middleware to submit the job to. Or in case of a programming error, it should just return the error to the client.

When a workflow is submitted, the interaction between the workflow components should be taken into account. The workflow components which require a lot of interaction should be scheduled on nearby resources. It would be an interesting future work to enable the interaction between jobs scheduled to different middleware systems.

In the current implementation there is a limit, which states how many jobs can be submitted concurrently per user. This was introduced, because the scheduling elements can be overwhelmed if they receive too many request concurrently. This limit should be changed to take into account the number of jobs submitted to a given scheduling element, not the number of jobs submitted by one user. A user can submit several jobs to several scheduling elements without reaching the limit alone. If multiple users submit multiple jobs to the same scheduling element in the same time, it can be that individually they won't reach the preset limit, but collectively would overload the WMS.

Another improvement could be made regarding the specification of the job description. In the current implementation for each attribute a SOAP call is sent to the server. A method could be implemented which would parse the whole job description and transmit it to the server using only one SOAP call. Regarding the job description, another

improvement can be made. The job description should be tested, to see that it contains all the required attributes, before a job is submitted.

Querying the status of a job is a heavy operation which should be performed as seldom as possible. Thus another improvement could involve finding an algorithm to reduce the number of such requests. This probably should query more frequently until the job start and then query only when the estimated job execution time is done. This includes predicting how much the queue wait time would be for a job.

All the remaining integration issues should be solved with the WS-VLAM system and the new Datafluo architecture [1] of WS-VLAM also need to be integrated with this module.

Implementing some of these features would improve the reliability and usability of the gMInION module .

[1] Datafluo is a dataflow approach with weaker scheduling constraints whereby each workflow task is scheduled dependant on data availability and hence alleviates co-allocation and encourages better resource usage.

Appendix A

Proxy certificates

A proxy certificate is very similar to a X.509 digital certificate, except that it's not signed by a CA ; it's signed by the end user. The purpose of the proxy certificate is to enable remote services to act on a user's behalf. Using a proxy prevents transferring the user's private key, facilitates single sign-on and it allows specifying a shorter lifetime, this way the encryption key associated with the proxy is less likely to be compromised within the validity time.

There are three types of Proxy Certificates in use by various versions of the GT .

Legacy proxy certificates were first introduced in GT 2.0 and care still supported by GT 4.x. These lack the ProxyCertInfo extension and the use of "CN=proxy" or "CN=limited proxy" distinguished name components. They can be generated in GT 4 through the use of 'grid-proxy-init -old'.
Proxy Draft Proxy Certificates (or "GSI3 Proxy Certificates" because their first appeared in GT 3) are certificates that are very similar to RFC 3820 with the exception that the ProxyCertInfo extension is identified with a non-standard object identifier . (Defined in the C code by PROXYCERTINFO_OLD_OID and in java by GSIConstants GSI_3_IMPERSONATION_PROXY). In GT 4.0.x Proxy Draft Proxy Certificates are the default with grid-proxy-init.

The third type are the RFC 3820 compliant proxies. In GT 4.0.x these can be generated using 'grid-proxy-init -rfc'. In GT 4.2.x, these can be generated by grid-proxy-init by default.

A VOMS proxy contains information about membership in particular VO s by embedding as an extension an Attribute Certificate, which is obtained from the VOMS . The VOMS manages information about the roles and privileges of users within a VO . The difference between a gLite and a Globus proxy is that instead of an entirely self-signed proxy, gLite uses so-called VOMS proxies for authentication and authorization. VOMS proxies are enhanced Globus proxies, that contain extensions signed by a VOMS server about the user's roles, group memberships and capabilities to simplify authorization at the Grid site level and spare the site administrators the task of updating huge user databases. In order to receive a VOMS proxy, users must contact a VOMS server in a connection secured with a Globus proxy and request certain roles, groups and capabilities. If the user is a member in the VO , that is administered by the VOMS server, this request will either be granted or denied based on the suitability of the user for the requested memberships. If the request is granted, the server will generate a signed certificate containing the granted request and store that in the user's proxy. The user can now impersonate him/herself at Grid sites belonging to the same VO as he/she does and will receive authorization compliant to the granted capabilities.

Appendix B

Adding VOMS attributes

```
private VOMSProxyInit m_proxyInit;
private VOMSRequestOptions m_requestOptions;

private static final int OID_OLD = 2;            // default
private static final int OID_RFC820 = 4;
private static final int DELEGATION_NONE = 1;
private static final int DELEGATION_LIMITED = 2;
private static final int DELEGATION_FULL = 3;    //default

private String serverN = "voms://voms.grid.sara.nl:30000"+
 "/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl";

public VomsInit(GlobusGSSCredentialImpl cred, String uservo)
 throws URISyntaxException, GSSException {
    URI uri = new URI(serverN);
    if (uri.getHost() != null) {
        VOMSServerInfo server = new VOMSServerInfo();
        server.setHostName(uri.getHost());
        server.setPort(uri.getPort());
        server.setHostDn(uri.getPath());
        server.setVoName(uservo);
        m_proxyInit =
 VOMSProxyInit.instance(((GlobusGSSCredentialImpl)
 cred).getGlobusCredential());
        m_proxyInit.addVomsServer(server);
        m_proxyInit.setProxyOutputFile(Util.proxyF);
        m_requestOptions = new VOMSRequestOptions();
        m_requestOptions.setVoName(uservo);

        int lifetime = cred.getRemainingLifetime() - 30;
        m_proxyInit.setProxyLifetime(lifetime);
        m_requestOptions.setLifetime(lifetime);

        m_proxyInit.setDelegationType(DELEGATION_FULL);
```

```
            m_proxyInit.setProxyType(OID_OLD);
        }
    }

    public GSSCredential createProxy() throws GSSException {
        // create
        GlobusCredential globusProxy;
        if ("NOVO".equals(m_requestOptions.getVoName())) {
            // TEST to create gridProxy :
            globusProxy = m_proxyInit.getVomsProxy(null);
        } else {
            ArrayList options = new ArrayList();
            options.add(m_requestOptions);
            globusProxy = m_proxyInit.getVomsProxy(options);
            // validate
            try {
                Vector v =
     VOMSValidator.parse(globusProxy.getCertificateChain());
                for (int i = 0; i < v.size(); i++) {
                    VOMSAttribute attr = (VOMSAttribute)
     v.elementAt(i);
                    if
     (!attr.getVO().equals(m_requestOptions.getVoName())) {
                        logger.fatal("The VO name of the created
     VOMS proxy ('" + attr.getVO() + "') does not match with the
     required VO name ('" + m_requestOptions.getVoName() + "').");
                    }
                }
            } catch (IllegalArgumentException iAE) {
                logger.fatal("The lifetime may be too long", iAE);
            }
        }
        return new GlobusGSSCredentialImpl(globusProxy,
     GSSCredential.INITIATE_AND_ACCEPT);
    }
```

To see what extensions the proxy files have I used the following command:

```
openssl x509 -text -certopt ext_parse < /tmp/proxyF
```

Appendix C

Setting up mutual authentication in Tomcat

This section describes how Tomcat can be integrated with SSL [66] to achieve mutual authentication using the grid certificates. The prerequisites are the following:

- a valid host certificate has to be present on the server side
- a valid user certificate or proxy certificate has to be present on the client machine (the machine which initiates the job submission request)
- a java keystore [1] and truststore [2] has to be set up on the client side
- both on the server and the client side the CA certificates, signing policies and certificate revocation lists files have to be installed (usually these can be found in the /etc/grid-security/certificates directory)

On the server side, where the Tomcat container is installed the following steps are needed:

1. Copy the host's private key and its certificate (hostkey.pem and hostcert.pem) in the Tomcat folder and change their ownership so that the Tomcat user is able to read them
2. Several libraries need to be copied to $CATALINA_HOME/lib folder in order to integrate Tomcat and SSL. These libraries can be obtained from installing the glite-security-trustmanager and glite-security-util-java [3] and they are the following:
   - glite-security-trustmanager.jar
   - glite-security-util-java.jar
   - log4j-1.2.8.jar
   - bcprov-jdk14-122.jar

[1] A keystore contains private keys, and the certificates with their corresponding public keys.

[2] A truststore is a keystore which is used when making decisions about what to trust. It contains certificates from other parties which one expects to communicate with, or from Certificate Authorities which are trusted to identify other parties.

[3] glite-security-trustmanager together with glite-security-util-java is an implementation of the java TrustManager interface with implementation of certificate path checking, grid name space restrictions and dynamic loading of CA certificates, credentials and certificate revocation lists. It can be used both in the server side for the server SSL handler and on the client side for the opening of SSL connections.

3. Edit the $CATALINA_HOME/conf/server.xml Tomcat configuration file, in order to enable Tomcat-SSL integration. A similar ¡Connector¿ element has to be present:

```
<Connector port="8443"
  maxThreads="150"
  minSpareThreads="25"
  maxSpareThreads="75"
  enableLookups="false"
  disableUploadTimeout="true"
  acceptCount="100"
  debug="0"
  scheme="https"
  SSLEnabled="true"
  clientAuth="true"
  sslProtocol="TLS"
  secure="true"
  log4jConfFile=
 "\$CATALINA_HOME/conf/log4j-trustmanager.properties"
  sslCAFiles="/etc/grid-security/certificates/*.0"
  crlFiles="/etc/grid-security/certificates/*.r0"
  sslKey= "\$CATALINA\_HOME/conf/hostkey.pem"
  sslCertFile= "\$CATALINA\_HOME/conf/hostcert.pem"
  SSLImplementation=
 "org.glite.security.trustmanager.tomcat.TMSSLImplementation"
 />
```

   In this file the paths for log4jConfFile, sslKey and sslCertFile should point to the appropriate files.

On the client side the truststore and keystore files should be set up.
Necessary steps to create the keystore using the user certificate:

1. Create keystore:

```
keytool -genkey -alias javaKS -keystore
 /home/tbalint/.globus/userKS.jks
```

2. Delete created keys:

```
keytool -delete -alias javaKS -keystore
 /home/tbalint/.globus/userKS.jks
```

3. Convert usercert.pem to pkcs12:

```
openssl pkcs12 -export -in usercert.pem -inkey userkey.pem -out
 usercert.p12
```

   (Usually the usercert.pem and userkey.pem are located in the $HOME/.globus directory.)

4. Import pkcs12 to Java keystore:

```
keytool -importkeystore -srckeystore usercert.p12 -destkeystore
 userKS.jks -srcstoretype pkcs12 -deststoretype jks
```

The following script can be used to create the truststore using all the trusted CA certificates, which in this case are located in the /etc/grid-security/certificates directory:

```
#!/bin/bash
GRID_SECURITY=/etc/grid-security
GRID_SECURITY_LOCAL=/home/tbalint/.globus
CA_JKS=${GRID_SECURITY_LOCAL}/caTruststore.jks

TMPFILE=$(mktemp) || exit 1

for i in ${GRID_SECURITY}/certificates/*.0 ; do
    j=$(echo $i | sed -e 's_.*/__g;s/\.0$//')
    cat $i | grep -A 100 "BEGIN CERTIFICATE-----" > $TMPFILE
    keytool -importcert -keystore $CA_JKS.$$ -noprompt
 -storepass password -alias $j -file $TMPFILE || echo -e
 "\n\tFAILED: $i\n"
done
mv -f $CA_JKS.$$ $CA_JKS
```

In the previous script the keystore password has to be modified.

After these steps, the client has to specify where the keystore and truststore are located and what are the passwords necessary to access these. In case of a client written in Java the following system properties need to be set:

```
System.setProperty("javax.net.ssl.keyStore", keyStore);
System.setProperty("javax.net.ssl.keyStorePassword",
 keyStorePassword);
System.setProperty("javax.net.ssl.trustStore", trustStore);
System.setProperty("javax.net.ssl.trustStorePassword",
 trustStorePassword);
```

Appendix D

Direct job submission to Globus and gLite grid middleware systems

D.1 Using the Globus Toolkit

To be able to run jobs on the grid a private certificate is needed, which binds together a person's identity and a public key, which can be used to identify the person in the digital world. This means that probably in the home directory there is a *.globus* directory, which contains the signed certificate (usercert.pem) and the private key (userkey.pem). A user also has to join a Virtual Organization to be able to use the resources.

After these steps a X.509 proxy certificate can be created using :

```
-bash-3.00$ grid-proxy-init
Your identity: /O=dutchgrid/O=users/O=nikhef/CN=Tunde Balint
Enter GRID pass phrase for this identity:
Creating proxy ..................... Done
Your proxy is valid until: Tue Nov 17 23:27:45 2009
```

Pre-WS GRAM (GT 2) used RSL to specify a job description. WS GRAM (GT 4) uses an xml based language for this same purpose, which can be obtained converting a pre-WS GRAM RSL file [1].

To submit a job *globusrun* or *globus-job-run* can be used. The *globusrun* command can be used to submit a job described in an rsl style. The *globus-job-submit* and *globus-job-run* commands translate the commands to a rsl-file which is given to *globusrun*. The rsl file can be seen using the -dumprsl option. *globus-job-run* command can be used to run a command on a local or remote machine as it was interactive. Using this command the job isn't submitted to the queuing system. *globus-job-submit* submits a job through a Globus Job Manager and it returns a URL specifying the job identifier. Getting the status of the submitted job can be done with the *globus-job-status* command and the output can be retrieved with *globus-job-get-output*

Testing the *globus-job-run* command:

```
-bash-3.00$ globus-job-run tbn14.nikhef.nl:2119/jobmanager
 /bin/date
Tue Nov 17 13:00:48 CET 2009
```

The RSL file generated when a simple job is submitted.

```
-bash-3.00$ globus-job-submit -dumprsl
  tbn14.nikhef.nl:2119/jobmanager -d /tmp -stdout output.txt
  /bin/date
&(executable="/bin/date")
(directory="/tmp")
(stdout="output.txt")
(stderr=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stderr
  anExtraTag)
```

Submitting the job, getting the status and trying to get the output:

```
-bash-3.00$ globus-job-submit  tbn14.nikhef.nl:2119/jobmanager
  -d /tmp -stdout output.txt /bin/date
  https://tbn14.nikhef.nl:20001/27577/1258460813/
-bash-3.00$ globus-job-status
  https://tbn14.nikhef.nl:20001/27577/1258460813/
DONE
-bash-3.00$ globus-job-get-output
  https://tbn14.nikhef.nl:20001/27577/1258460813/
Invalid job id.
```

The output cannot be obtained in this case with the *globus-job-get-output* command, since the job already finished and it wrote the output to the specified file. In order to obtain the output of the job, we need to copy the specified 'stdout' file with GridFTP. This can be done in the following way:

```
-bash-3.00$ globus-url-copy
  gsiftp://tbn14.nikhef.nl/tmp/output.txt  file:$HOME/output.txt
```

If this command executes successfully the output of the job can be seen in the $HOME directory.

Staging in and out files is done the following way:

```
-bash-3.00$ globus-job-run  -dumprsl
  tbn14.nikhef.nl:2119/jobmanager -stdin -s input.txt -stdout
  -s output.txt -d /tmp /bin/cat
   &(executable="/bin/cat")
     (directory="/tmp")
     (stdin=$(GLOBUSRUN_GASS_URL) # "/user/tbalint/input.txt")
     (stdout=$(GLOBUSRUN_GASS_URL) # "/user/tbalint/output.txt")
```

After running the previous RSL , the content of input.txt is copied to output.txt. Since the output.txt is also staged, we don't have to retrieve it separately.

D.2 Using gLite

First of all in your home directory you should have a .globus directory with the signed certificate (usercert.pem) and your private key (userkey.pem).

To be able to identify to whom a job belongs a proxy certificate has to be delegated with every job. This also assures that access is granted to the resources that the user is entitled to. To obtain a short term proxy you need to log in to the User Interface and use the *voms-proxy-init* command.

```
-bash-3.00$ voms-proxy-init --voms pvier
Cannot find file or dir: /user/tbalint/.glite/vomses
Enter GRID pass phrase:
Your identity: /O=dutchgrid/O=users/O=nikhef/CN=Tunde Balint
Creating temporary proxy ..... Done
Contacting  voms.grid.sara.nl:30000
 [/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl]
 "pvier" Done
Creating proxy ..... Done
Your proxy is valid until Fri Nov 20 00:31:58 2009
```

Submitting a job to the gLite Workload Management System

Submitting and managing a job using the WMS can be done using the following steps:

1. Delegate the valid proxy credential to the WMS, which is responsible of accepting incoming request from the UI. This can be done either explicitly with a separate command, which generates a delegation ID:

   ```
   -bash-3.00$ glite-wms-job-delegate-proxy -d $USER
   Connecting to the service
    https://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
   ==================== glite-wms-job-delegate-proxy Success ========
   Your proxy has been successfully delegated to the WMProxy:
   https://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
   with the delegation identifier: tbalint
   ================================================================
   ```

   or automatically, when each operation contains the -a option specifying that delegation is needed.

2. Create a JDL file:

   ```
   -bash-3.00$ cat Host.jdl
   Executable = "/bin/hostname";
   Arguments = "-f";
   Stdoutput = "host.txt";
   StdError = "stderror.err";
   OutputSandbox = {"host.txt","stderror.err"};
   ```

   In this simple JDL file there are no requirements specified, only the standard output and standard error are redirected and the executable is indicated. The OutputSandbox attribute indicates the files to be copied back after job execution.

3. Submit the job

   ```
   -bash-3.00$ glite-wms-job-submit -d $USER -o myjobs Host.jdl
   Connecting to the service
    https://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
   ==================== glite-wms-job-submit Success ================
   The job has been successfully submitted to the WMProxy
   Your job identifier is:
   https://grasveld.nikhef.nl:9000/dDfhxdS-WZddvSDxEciBaA
   The job identifier has been saved in the following file:
   ```

```
/user/tbalint/exercises/Date/myjobs
```

The delegation ID has to be specified and the -o option is used to save the identifier of the job.

4. Check status

```
-bash-3.00$ glite-wms-job-status -i myjobs
```

5. And when the job is done, the output can be retrieved

```
-bash-3.00$ glite-wms-job-output -i myjobs --dir ./output/
```

In this case too, the delegation ID has to be specified and the –dir option indicates where the retrieved files should be saved.

Submitting a job to the LCG-CE

Submitting to the LCG-CE requires the same steps and commands as submitting a job using the Globus toolkit.

Obtain the job description which is going to be submitted:

```
-bash-3.2$ globus-job-submit -dumprsl
  ce.gina.sara.nl:2119/jobmanager -d /tmp -stdout -s output.txt
  -stderr -s error.txt /bin/hostname
   &(executable="/bin/hostname")
    (directory="/tmp")
    (stdout=$(GLOBUSRUN_GASS_URL) # "/user/tbalint/output.txt")
    (stderr=$(GLOBUSRUN_GASS_URL) # "/user/tbalint/error.txt")
```

Submit the job:

```
-bash-3.2$ globus-job-submit ce.gina.sara.nl:2119/jobmanager -d
  /tmp -stdout -s output.txt -stderr -s error.txt /bin/hostname
https://ce.gina.sara.nl:20018/20959/1286041647/
```

Obtain the status:

```
-bash-3.2$ globus-job-status
  https://ce.gina.sara.nl:20018/20959/1286041647/
DONE
```

Obtain the results:

```
-bash-3.2$ globus-job-get-output
  https://ce.gina.sara.nl:20018/20959/1286041647/
Invalid job id.
```

The output cannot be obtained in this case with the *globus-job-get-output* command, since the job already finished and it wrote the output to the specified file. In order to obtain the output of the job, we need to copy the specified 'stdout' file with GridFTP. This can be done in the following way:

```
-bash-3.00$ globus-url-copy
  gsiftp://ce.gina.sara.nl/tmp/output.txt file:$HOME/output.txt
```

If this command executes successfully the output of the job can be seen in the $HOME directory.

Submitting a job to the CREAM CE

To submit and manage a job to the CREAM CE the following steps are necessary:

1. Delegate the credentials to the chosen computing element:

```
-bash-3.2$ glite-ce-delegate-proxy -e stremsel.nikhef.nl tbalint
2010-10-02 20:00:08,196 WARN - No configuration file suitable
 for loading. Using built-in configuration
2010-10-02 20:00:09,353 NOTICE - Proxy with delegation id
 [tbalint] succesfully delegated to endpoint
 [https://stremsel.nikhef.nl:8443/
 /ce-cream/services/gridsite-delegation]
```

2. Create the job description file. We created a simple job description file which interrogates the name of the host on which the job runs. The OutputSandboxBaseURI has to be specified, otherwise we won't know from where the job output can be retrieved back to the client:

```
-bash-3.2$ cat host.jdl
[
Executable = "/bin/hostname";
Arguments = "-f";
StdOutput = "output.txt";
StdError = "error.txt";

OutputSandbox = {
                    "output.txt",
                    "error.txt"
                };

OutputSandboxBaseDestURI = "gsiftp://stremsel.nikhef.nl/tmp";
]
```

3. Submitting the job. The delegation identifier has to be specified or the proxy can be automatically delegated. If automatic delegation is used, the proxy is delegated for every job separately.

```
-bash-3.2$ glite-ce-job-submit -r
 stremsel.nikhef.nl/cream-pbs-short --delegationId tbalint
 host.jdl
2010-10-02 20:11:15,638 WARN - No configuration file suitable
 for loading. Using built-in configuration
https://stremsel.nikhef.nl:8443/CREAM697572024
```

4. Obtaining the status of the job:

```
-bash-3.2$ glite-ce-job-status
 https://stremsel.nikhef.nl:8443/CREAM697572024
2010-10-02 20:11:36,095 WARN - No configuration file suitable
 for loading. Using built-in configuration
******  JobID=[https://stremsel.nikhef.nl:8443/CREAM697572024]
        Status          = [DONE-OK]
        ExitCode        = [0]
```

5. Obtain a list of the jobs belonging to a user, which are running on a specified computing element:

```
-bash-3.2$ glite-ce-job-list stremsel.nikhef.nl
2010-10-02 20:11:46,700 WARN - No configuration file suitable
  for loading. Using built-in configuration
https://stremsel.nikhef.nl:8443/CREAM697572024
```

6. Obtain the output files:

```
-bash-3.2$ globus-url-copy
  gsiftp://stremsel.nikhef.nl/tmp/output.txt file:$HOME/output
```

7. Purge the job, to clean up the working directory oncomputing element:

```
-bash-3.2$ glite-ce-job-purge
  https://stremsel.nikhef.nl:8443/CREAM697572024
2010-10-02 20:15:47,750 WARN - No configuration file suitable
  for loading. Using built-in configuration

Are you sure you want to purge specified job(s) [y/n]: y
```

Appendix E

JavaGAT

The GAT specification has a three-tier design consisting of GAT-API, GAT Engine, and GAT Adaptors. GAT-API defines a simple, platform-independent API to generic Grid resources and services. The GAT engine is a runtime library which provides the function bindings for the GAT-API. It is responsible of the selection and loading of the adaptors depending on the client request, required properties and grid service availability. The GAT Engine consists of three logical parts:

1. The objects providing the GAT API functions, which map the API function to the adaptor-specific functionality
2. The adaptor management subsystem, which is responsible of loading the adaptors, managing their lifetime and maintaining a capability registry which allows the selection of the right adaptor. Each adaptor is registered to this capability registry and every capability of the adaptor has an attached meta data (called 'preferences') to allow easier selection.
3. Utility objects and functions for error handling and reporting.

The GAT engine provides decoupling by exposing two sets of APIs, one for the application and another one for the adaptor connection. The interface between the GAT engine and the adaptors is called the Capability Provider Interface, which mirrors the GAT API itself. An adaptor is compiled against the GAT engine and linked as a shared library. On loading the adaptor registers its capabilities.

The GAT-API is divided into several subsystems such as resource management, data management, event management, and information management. GAT adaptors can be divided into several types taking into account their functionality: File Management, FileStream Management, LogicalFile Management, Advertisable Management, Resource Management, Interprocess Communication, Job Management, and Monitoring.

Bindings for the GAT API exist for C, C++, Python and Java.

In the JavaGAT implementation adaptors are available for the following: SSH, gLite (experimental), Globus, GridSAM, local job (for testing mostly), SFTP Trilead, SSH Trilead SGE, Unicore.

When an application creates a GAT object using the GAT object factory, the engine must determine which adaptors are applicable for the requested object, and it instantiates the ones that are applicable. Java's reflection is used to lookup and invoke the constructor for the adaptors which fulfilled the requested preferences. The applicable adaptors are sorted taking into account performance or security. The user can influence the sorting by specifying an adaptor ordering policy. Next JavaGAT creates a proxy class

(using *java.lang.reflect.Proxy*) that implements the interface of the requested object, but forwards the application's calls to one or more adaptors.

JavaGAT also provides a mechanism to create a sandbox on the remote machines, and to pre- and post-stage files. For file access, JavaGAT has adaptors for local files, GridFTP, RFT, FTP, SSH, SFTP, HTTP, HTTPS and SMB/CIFS. For resource management it supports local forking, GRMS, Globus GRAM, gLite WMS, SSH, prun, PBS, Sun Grid Engine, ProActive, Integrade and Zorilla.

The JavaGAT security context is used to provide security information regarding passwords, certificates and we can even retrieve credentials from the MyProxy credential manager.

JavaGAT can be downloaded from:
`http://gforge.cs.vu.nl/gf/project/javagat/frs/`.

The following tests were done with version 2.0.5. The prerequisites of installing JavaGAT are Java SUN JDK version 6 or newer and Ant. The archive has to be extracted to a directory, the environmental variable *$GAT_LOCATION* has to be set to the directory where JavaGAT should be installed, and then by typing ***ant*** in the directory where the archive was extracted the program is installed.

E.1 Security

To create a proxy certificate you can use the grid-proxy-init of GAT, which is located in $GAT_LOCATION/bin. The script invokes the class *org.globus.tools.ProxyInit* which is a class of the Java Cog Kit, and delivered with GAT. This class reads in some configuration information from the dataset $HOME/.globus/cog.properties, but only if this file is available. *cog.properties* might look as follows:

```
usercert=/home/tunde/.globus/usercert.pem
userkey=/home/tunde/.globus/userkey.pem
proxy=/tmp/x509up_u1000
cacert=/etc/grid-security/certificates
```

The parameters have the following meaning:

1. usercert: The path to the file containing your grid certificate.
2. userkey: The path to the file containing your grid key.
3. proxy: The name under which your proxy certificate which you create with grid-proxy-init is stored
4. cacert: The path of the directory, which contains the host certificates

Without any configurations in cog.properties, the required files will be searched for in their default locations according to the following rules:

1. If the usercert property is not set, the $X509_USER_CERT system property is checked. If the system property is not set, the value defaults to $HOME/.globus/usercert.pem.
2. If the userkey property is not set, the $X509_USER_KEY system property is checked. If the system property is not set, the value defaults to $HOME/.globus/userkey.pem.

3. If the cacert property is not set, first the $X509_CERT_DIR system property is checked. If the system property is not set, then the $HOME/.globus/certificates directory is checked. If the directory does not exist, and on a Unix/Linux machine, the directory with the name /etc/grid-security/certificates is checked next. If one of these directories with certificates is found, all the certificates in that directory will be loaded and used.

4. If the proxy property is not set, first the $X509_USER_PROXY system property is checked. If the system property is not set, then it defaults to a value based on the following rules: If a UID system property is set, and running on a Unix/Linux machine it returns /tmp/x509up_u$UID. If any other machine then Unix/Linux, it returns $tempdir/x509up_u$UID, where tempdir is a platform specific temporary directory as indicated by the java.io.tmpdir system property. If a UID system property is not set, the user name will be used instead of the UID. That is, it returns $tempdir/x509up_u$username.

E.2 Running programs

It is assumed that all the source files are in a *src* directory. Using Apache Ant a JAR file is created including the GAT-engine.jar. The used *build.xml* file looks like this:

```
<project name="JavaGAT Tests" default="usage" basedir=".">
  <property environment="env"/>
  <property name="srcdir"    location="src"/>
  <property name="builddir" location="build"/>
  <property name="distdir"   location="jars"/>
  <property name="docdir"    location="docs"/>
    <property name="cpath"
 location="${env.GAT_LOCATION}/lib/GAT-engine.jar"/>
  <target name="usage" description="Print usage string">
    <echo message="JavaGAT Tests"/>
    <echo message="  ant build  : build the test jarfile."/>
    <echo message="  ant clean  : to clean the tree."/>
  </target>
  <target name="check-environment">
    <condition property="ant.correct">
      <isset property="env.ANT_HOME"/>
    </condition>
    <condition property="gat.set">
      <isset property="env.GAT_LOCATION"/>
    </condition>
    <available file="${cpath}" property="gat.correct"/>
  </target>
  <target name="check-gat-correct" unless="gat.correct">
    <echo message="Your $GAT_LOCATION is not set correctly
 (file not found)!"/>
    <fail/>
  </target>
  <target name="check-gat-set" unless="gat.set">
    <echo message="Your $GAT_LOCATION is not set!"/>
    <fail/>
```

```
  </target>
  <target name="check-ant-correct" unless="ant.correct">
    <echo message="Your $ANT_HOME is not set!"/>
    <fail/>
  </target>
  <target name="prepare">
    <mkdir dir="${distdir}" />
    <mkdir dir="${builddir}" />
  </target>
  <target name="perform-build" depends="prepare">
    <delete failonerror="false" file="${distdir}/test.jar" />
    <javac srcdir="${srcdir}"
      destdir="${builddir}"
      includes="**/**"
      classpath="${cpath}"
      debug="true"
      deprecation="true">
      <compilerarg value="-Xlint"/>
    </javac>
    <jar jarfile="${distdir}/test.jar">
      <fileset dir="${builddir}" >
        <include name="**/**" />
      </fileset>
    </jar>
    <delete failonerror="false" dir="${builddir}" />
  </target>
  <target name="build"
 depends="check-environment,check-ant-correct,
 check-gat-set,check-gat-correct,prepare,perform-build"
 description="Build the test jar file"/>
  <target name="clean"
     description="Clean the tree">
    <delete failonerror="false" dir="${distdir}" />
    <delete failonerror="false" dir="${builddir}" />
  </target>
</project>
```

After creating the test.jar file in the *jars* directory, making sure that the $GAT_LOCATION is set and that the adaptors can be found, the programs can be run. To make the execution of the programs and the setting of the classpath easier, a script was created:

```
tunde@schrift:~/JavaGAT/JavaGAT-2.0.5/javaGat_tests\$ cat
 runPrg.sh
#!/bin/sh
if [ -z "$GAT_LOCATION" ] ; then
  echo GAT_LOCATION variable not set, using $PWD/..
  GAT_LOCATION=$PWD/..
fi

GAT_ENGINE_LOCATION=$GAT_LOCATION/lib
GAT_ADAPTOR_LOCATION=$GAT_LOCATION/lib/adaptors
```

```
add_to_gat_classpath () {
    DIRLIBS=${1}/*.jar
    for i in ${DIRLIBS}
    do
      if [ "$i" != "${DIRLIBS}" ] ; then
        if [ -z "$GAT_CLASSPATH" ] ; then
          GAT_CLASSPATH=$i
        else
          GAT_CLASSPATH="$i":$GAT_CLASSPATH
        fi
      fi
    done
}
add_to_gat_classpath $GAT_ENGINE_LOCATION
GAT_CLASSPATH=$GAT_CLASSPATH:$GAT_ENGINE_LOCATION
java -cp :./jars/test.jar:$GAT_CLASSPATH
 -Dgat.adaptor.path=$GAT_ADAPTOR_LOCATION
 -Dlog4j.configuration=$GAT_LOCATION/log4j.properties $*
```

E.3 Transferring Files

The URI semantics in JavaGAT is a bit different compared to java.net.URI. Using a full URI is easy: *protocol:///machine/path_to_file*, but sometimes some field can be left blank and then the number of / counts. For example:

1. file:///output - means a local file in the current directory
2. file:////output - means a local file in the root (/) directory
3. file:////tmp/output - means a local file in the /tmp directory
4. ftp://10.0.0.1/output - means a remote file in default ftp directory
5. gsiftp://tbn14.nikhef.nl//tmp/jGAT1.txt - means a remote file in the /tmp directory

With JavaGAT late binding can be used by specifying *any://* as the protocol in the URI, otherwise early binding can be used by forcing a specific adaptor, choosing one of the following protocols: *ftp://, gsiftp://, http://, file://,...*
To transfer a file in the *src* directory a simple program was created, which takes as arguments the source and the destination from/to where the file should be copied:

```
tunde@schrift:~/JavaGAT/JavaGAT-2.0.5/javaGat_tests$ cat
 src/FileCopy.java
import org.gridlab.gat.GAT;
import org.gridlab.gat.URI;
import org.gridlab.gat.io.File;
class FileCopy {
  public static void main(String[] args) {
    if (args.length != 2)
    {
      System.err.println("Usage: ");
```

```
      System.err.println("runPrg.sh FileCopy src_url dest_url");
    } else {
      try {
        GAT.createFile(new URI(args[0])).copy(new URI(args[1]));
        System.err.println("OK");
        GAT.end();
      } catch (Exception e) {
        System.err.println("Failed: " + e);
        e.printStackTrace();
      }
    }
  }
}
```

After running *ant build*, we can copy file locally and even using Globus or gLite, if the necessary proxy certificate is already generated. If we didn't create the proxy file, then by specifying a *CertificateSecurityContext* in the program, we can still transfer the file. In this case the program has to be modified in the following way:

```
import org.gridlab.gat.GAT;
import org.gridlab.gat.URI;
import org.gridlab.gat.io.File;
import org.gridlab.gat.GATContext;
import org.gridlab.gat.security.CertificateSecurityContext;
import java.io.Console;

class FileCopy {
  public static void main(String[] args) {
    if (args.length != 2)
    {
      System.err.println("Usage: ");
      System.err.println("runPrg.sh FileCopy src_url dest_url");
    } else {
      try{
        char[] passwd = null;
        Console cons;
        if ((cons = System.console()) != null && (passwd =
 cons.readPassword("[s]", "Enter password for the
certificate:") != null) {}
        String userPass = new String(passwd);
        GATContext context = new GATContext();
        CertificateSecurityContext secContext = new
 CertificateSecurityContext(
          new URI("/home/tunde/.globus/userkey.pem"),
          new URI("/home/tunde/.globus/usercert.pem"),
          userPass);
        context.addSecurityContext(secContext);
        GAT.createFile(context,new URI(args[0])).copy(new
 URI(args[1]));
        System.err.println("OK");
        GAT.end();
      } catch (Exception e) {
```

```
            System.err.println("Failed: " + e);
            e.printStackTrace();
        }
    }
  }
}
```

Running

```
./runPrg.sh FileCopy file:////tmp/javagat_test.txt
  gsiftp://tbn14.nikhef.nl//tmp/javagat_test.txt
```

will result in successfully creating a globus proxy and transferring the file.

The GATContext class represents the state and security context of an application. It is used to encapsulate a number of GAT API method calls into a common scope, including adaptor loading preferences, security context and status code management. Several security context can be attached to one GATContext, this way the GAT will use the one with which it can fulfill the request.

The SecurityContext class stores security information for authentication and authorization.

To create a gLite security context additional information is needed, since a VOMS proxy has to be used. The following data is required:

1. The name of the VO for which the user wants to obtain a credential
2. The endpoint of the VOMS server webservice
3. The port at which the VOMS server is listening to requests
4. The distinguished name (DN) of the VOMS Host.

To create a VOMS proxy the following properties need to be added to the security context:

```
Preferences globalPrefs = new Preferences();
globalPrefs.put("VirtualOrganisation", "pvier");
globalPrefs.put("vomsHostDN",
 "/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl");
globalPrefs.put("vomsServerURL", "voms.grid.sara.nl");
globalPrefs.put("vomsServerPort", "30000");
context.addPreferences(globalPrefs);
```

According to the documentation additional information can be provided, like the desired proxy lifetime (in seconds) setting the *vomsLifeTime* property.

It is possible to use a previously created VOMS proxy certificate. If the glite.reuseProxy system property is set to true, the system will check, whether a valid VOMS proxy exists. If such a proxy is found, it is determined, whether the proxy lifetime exceeds the one specified in the vomsLifetime property. If the vomsLifetime property is unspecified, it is checked whether the proxy is still valid for more than 10 minutes. The following code has to be added to the source code (more specifically some global preferences need to be added to the security context):

```
Preferences globalPrefs = new Preferences();
globalPrefs.put("glite.reuseProxy","true");
context.addPreferences(globalPrefs);
```

Regarding the gLite file management it is stated that SRM and GUID are supported, but these are still experimental. LFN is not supported. Supported operations are: copy and delete both for SRM and GUID; creating a new file is only possible with GUID.

To specify that the SRM adaptor should be used, another property has to be added:

```
globalPrefs.put("File.adaptor.name","glitesrm");
```

Then running

```
./runPrg.sh FileCopy file:////tmp/test\_SRM.txt
 srm://tbn18.nikhef.nl:8446
 //dpm/nikhef.nl/home/pvier/tbalint/t.txt
```

will copy the file to the SRM.

E.4 Job submission

To submit a job, the description has to be 'translated' into the job description language used by the underlying middleware. The create a job submission a software and a hardware description has to be created. Meaning that the submitted job has to be described and than the hardware requirements and the software requirements need to be specified. In case of a globus job, it can be run with a previously generated proxy certificate, and than no security context has to be specified. If there is no proxy generated, a job can be submitted in the following way:

```
import java.util.Hashtable;
import org.gridlab.gat.*;
import org.gridlab.gat.io.*;
import org.gridlab.gat.resources.*;
import org.gridlab.gat.security.CertificateSecurityContext;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;


public class GlobusJob {
  private static String getPassphrase() {
    JPasswordField pwd = new JPasswordField();
    Object[] message = { "grid-proxy-init\n Please enter your
 passphrase.",pwd };
    JOptionPane.showMessageDialog(null, message,
 "Grid-Proxy-Init", JOptionPane.QUESTION_MESSAGE);
    return new String(pwd.getPassword());
  }
  public static void main(String [] args) throws Exception {
    String userPass = getPassphrase();
    GATContext context = new GATContext();
    CertificateSecurityContext secContext = new
 CertificateSecurityContext(
      new URI("/home/tunde/.globus/userkey.pem"),
      new URI("/home/tunde/.globus/usercert.pem"),
      userPass);
    context.addSecurityContext(secContext);
    Preferences prefs = new Preferences();
```

```
        prefs.put("ResourceBroker.adaptor.name", "globus");
        context.addPreferences(prefs);
        File stdout = GAT.createFile(context,"any:///stdout");
        File stderr = GAT.createFile(context,"any:///stderr");
        SoftwareDescription sd = new SoftwareDescription();
        sd.setExecutable("/bin/hostname");
        sd.setStdout(stdout);
        sd.setStderr(stderr);
        sd.setArguments(new String [] { "-f" });
        ResourceBroker broker = GAT.createResourceBroker(context,
    new URI("any://"+args[0]+"/jobmanager"));
        ResourceDescription rd = new
    HardwareResourceDescription(new Hashtable<String,Object>());
        JobDescription jd = new JobDescription(sd, rd);
        Job job = broker.submitJob(jd);
        Job.JobState state = job.getState();
        while (state != Job.JobState.STOPPED && state !=
    Job.JobState.SUBMISSION_ERROR) {
          try {
            System.out.println("Sleeping!");
            Thread.sleep(1000);
          } catch (Exception e) { // ignore }
          state = job.getState();
        }
        if (state == Job.JobState.SUBMISSION_ERROR) {
          System.out.println("ERROR");
        } else {
          System.out.println("OK");
        }
        GAT.end();
    }
}
```

Then we can build it with ant and run it using:

```
./runPrg.sh GlobusJob tbn14.nikhef.nl:2119
```

The SoftwareDescription specifies the basic detail of a job, like the location of input and output files, command line arguments, environmental variables and how to handle stdin, stdout, stderr. The HardwareResourceDescription contains a set of requirements which must be met by the hardware on which the job will run (e.g. required amount of memory or disk space). Dealing with executables in a grid environment can be done in two ways:

1. compile the application before submission and move the executable to the remote site
2. submit a shell script to the remote machine which downloads the source code and builds the application

In both cases a set of libraries must be installed on the machine and in case of the second scenario also a compiler has to be available to build the application. These constraints can be specified with the SoftwareResourceDescription.

The ResourceBroker is responsible for all interactions with the resources. It allows to search for resources, to reserve them for job submission and to submit jobs to it. It

is possible to specify which resource broker to use, for example in the previous example the following is stated: *prefs.put("ResourceBroker.adaptor.name", "globus");*

When trying to run a gLite job extra properties need to be set:

```
import java.util.Hashtable;
import org.gridlab.gat.*;
import org.gridlab.gat.io.*;
import org.gridlab.gat.resources.*;
import org.gridlab.gat.security.CertificateSecurityContext;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;


public class GliteJob {
  private static String getPassphrase() {
    JPasswordField pwd = new JPasswordField();
    Object[] message = { "grid-proxy-init\n Please enter your
 passphrase.", pwd };
    JOptionPane.showMessageDialog(null, message,
 "Grid-Proxy-Init", JOptionPane.QUESTION_MESSAGE);
    return new String(pwd.getPassword());
  }
  public static void main(String [] args) throws Exception {
    String userPass = getPassphrase();
    GATContext context = new GATContext();
    CertificateSecurityContext secContext = new
 CertificateSecurityContext(
      new URI("/home/tunde/.globus/userkey.pem"),
      new URI("/home/tunde/.globus/usercert.pem"),
      userPass);
    Preferences globalPrefs = new Preferences();
    globalPrefs.put("VirtualOrganisation", "pvier");
    globalPrefs.put("vomsHostDN",
"/O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl");
        globalPrefs.put("vomsServerURL", "voms.grid.sara.nl");
        globalPrefs.put("vomsServerPort", "30000");
        globalPrefs.put("gridftp.authenticate.retry","1");
    globalPrefs.put("ResourceBroker.adaptor.name", "glite");
        globalPrefs.put("File.adaptor.name", "GridFTP");

        globalPrefs.put("glite.reuseProxy","true");
        globalPrefs.put("gglite.deleteJDL", "false");

    context.addPreferences(globalPrefs);
    context.addSecurityContext(secContext);

    File stdout = GAT.createFile(context, "any:///stdout");
    File stderr = GAT.createFile(context, "any:///stderr");

    SoftwareDescription sd = new SoftwareDescription();
```

```
    sd.setExecutable("/bin/hostname");
        sd.setStdout(stdout);
        sd.setStderr(stderr);
    sd.setArguments(new String [] { "-f" });

    ResourceBroker broker = GAT.createResourceBroker(
 context,globalPrefs, new
 URI("any://graspol.nikhef.nl:7443/glite_wms_wmproxy_server"));

    ResourceDescription rd = new
 HardwareResourceDescription(new Hashtable<String,Object>());
    JobDescription jd = new JobDescription(sd, rd);
    Job job = broker.submitJob(jd);
    Job.JobState state = job.getState();
    while (state != Job.JobState.STOPPED && state !=
 Job.JobState.SUBMISSION_ERROR) {
      try {
        System.out.println("Sleeping!");
        Thread.sleep(1000);
      } catch (Exception e) {
        // ignore
      }
      state = job.getState();
    }
    if (state == Job.JobState.SUBMISSION_ERROR) {
      System.out.println("ERROR");
    } else {
      System.out.println("OK");
    }
    GAT.end();
  }
}
```

In case there is a VOMS proxy available I got the following error:

```
18:17:57 ERROR [main]
 org.glite.security.trustmanager.ContextWrapper
  - Rejecting a CRL from CN=SwissSign Server Bronze CA,
  E=bronze@swisssign.com,O=SwissSign,C=CH because corresponding
  ca not found or invalid signature
18:17:57 ERROR [main]
 org.glite.security.trustmanager.ContextWrapper
  - Rejecting a CRL from CN=SwissSign Personal Bronze CA,
  E=bronze@swisssign.com,O=SwissSign,C=CH because corresponding
  ca not found or invalid signature
18:17:57 ERROR [main]
 org.glite.security.trustmanager.ContextWrapper
  - Rejecting a CRL from C=CH,O=SwissSign,CN=SwissSign
  Personal Bronze CA,E=bronze@swisssign.com because
  corresponding ca not found or invalid signature
18:17:58 ERROR [main]
 org.glite.security.trustmanager.ContextWrapper
```

```
 - Rejecting a CRL from C=CH,O=SwissSign,CN=SwissSign Server
 Bronze
   CA,E=bronze@swisssign.com because corresponding ca
   not found or invalid signature
18:17:58 ERROR [main]
 org.gridlab.gat.resources.cpi.glite.GliteJob
 - Problem while copying input files
AxisFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}
         Server.userException
 faultSubcode:
 faultString: javax.net.ssl.SSLHandshakeException:
 Received fatal alert: handshake_failure
 faultDetail:
        {http://xml.apache.org/axis/}stackTrace:
   javax.net.ssl.SSLHandshakeException:
      Received fatal alert: handshake_failure
.....
   at org.gridsite.www.namespaces.delegation_1
      .DelegationSoapBindingStub.getProxyReq
    (DelegationSoapBindingStub.java:180)
```

After updating the CRL files, only the handshake failure remained:

```
13:17:48 ERROR [main]
 org.gridlab.gat.resources.cpi.glite.GliteJob
 - Problem while copying input files
AxisFault
--- START OF NESTED EXCEPTION ---
*** GliteResourceBrokerAdaptor failed (AxisFault): ; nested
 exception is:
   javax.net.ssl.SSLHandshakeException:
   Received fatal alert: handshake_failure
--- END OF NESTED EXCEPTION ---)
13:10:00 INFO  [main]
 org.gridlab.gat.engine.AdaptorInvocationHandler
 - invoke: No adaptor could be invoked.
AxisFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}
  Server.userException
 faultSubcode:
 faultString: javax.net.ssl.SSLHandshakeException:
 Received fatal alert: handshake_failure
 faultDetail:
        {http://xml.apache.org/axis/}stackTrace:
   javax.net.ssl.SSLHandshakeException:
      Received fatal alert: handshake_failure
.....
   at org.gridsite.www.namespaces.delegation_1.
      DelegationSoapBindingStub.getProxyReq
    (DelegationSoapBindingStub.java:180)
```

With no proxy was generated I got the following error:

```
11:44:39 DEBUG [main]
 org.gridlab.gat.engine.AdaptorInvocationHandler
- adaptor instantiation: GliteResourceBrokerAdaptor for type
ResourceBroker SUCCESS
VOMS Proxy generation...
18:59:14 INFO  [main]
 org.gridlab.gat.security.glite.GliteSecurityUtils
- Checking whether the VOMS proxy extensions correspond to the
JavaGAT preferences.
18:59:15 INFO  [main]
 org.gridlab.gat.security.glite.GliteSecurityUtils
- Current VOMS proxy extensions doesn't correspond
to the JavaGAT preference.
  Creation of a new proxy
18:59:15 INFO  [main]
 org.gridlab.gat.security.glite.GliteSecurityUtils
- Reason:
18:59:15 INFO  [main]
 org.gridlab.gat.security.glite.GliteSecurityUtils
  - Creating new VOMS proxy with lifetime (seconds): 43200
18:59:15 DEBUG [main]
 org.gridlab.gat.security.glite.GlobusProxyManager
  - creating gss name with host-dn
  /O=dutchgrid/O=users/O=nikhef/CN=Tunde Balint
18:59:15 DEBUG [main]
 org.gridlab.gat.security.glite.GlobusProxyManager
  - received voce voms gss socket at:
 voms.grid.sara.nl/145.100.13.52
18:59:17 INFO  [main]
 org.gridlab.gat.engine.AdaptorInvocationHandler
- ResourceBroker
(any://graszode.nikhef.nl:7443/glite_wms_wmproxy_server)
.submitJob ( JobDescription(softwareDescription:
 SoftwareDescription(executable: /bin/hostname, arguments:
 {null}, stdin: null, stdout: null, stderr: null,
 environment: null, preStaged: {}, postStaged: {}, attributes:
 {}),
 resourceDescription: ResourceDescription(attributes: {},
  resourceDescriptions: null), resource: null)) ->
  GliteResourceBrokerAdaptor FAILED
 (java.lang.AbstractMethodError:
   org.apache.xerces.dom.DocumentImpl.getXmlStandalone()Z)
18:59:17 INFO  [main]
 org.gridlab.gat.engine.AdaptorInvocationHandler
- invoke: No adaptor could be invoked.
Exception in thread "main" --- START OF NESTED EXCEPTION STACK
 TRACE ---
*** stack trace of AbstractMethodError
java.lang.AbstractMethodError:
 org.apache.xerces.dom.DocumentImpl.getXmlStandalone()Z
...
```

This error is reported usually when parsing an XML file with *javax.xml.transform.Transformer*. Workaround: downgrade java to version 1.5. Running the GliteJob program actually works with Java 1.5, but not with Java 1.6. The generated proxy file is saved in */tmp*. Also the previous Axis error regarding delegation disappeared. Even transferring the standard error and output files worked.

There are some gLite adaptor specific properties:

1. glite.pollIntervalSecs - how often should the job lookup thread poll the WMS for job status updates (value in seconds, default 3 seconds)

2. glite.deleteJDL - if this is set to true, the JDL ?le used for job submission will be deleted when the job is done. Default value is false.

3. glite.newProxy - if this is set to true, a new proxy is created even if the lifetime of the old one is still sufficient

The attributes for the software description are derived from the globus job submission file format (RSL). There are also some JDL specific attributes, like hostCount, which are supported as GLUE requirements. To specify GLUE requirements which cannot be set by the SoftwareResourceDescription or by the HardwareResourceDescription keys, the *glite.other* property can be set either for a SoftwareResourceDescription or for a HardwareResourceDescription and then a legal GLUE requirement can be set as entry, like:

!(other.GlueCEUniqueID == "some_ce_of_your_choice"). There is almost no feedback while trying several adaptors, so after a while we that the program only hangs. This can be solved by adjusting the debugging level in *$GAT_LOCATION/log4j.properties*.

Appendix F

SAGA API details

The SAGA API consists of two major components: the non-functional SAGA Look-&-Feel, and a set of functional SAGA packages. Under Look-&-Feel those SAGA classes and methods are listed, which determine (syntactic and semantic) aspects of the API which are mostly invariable over the set of functional API packages, such as error handling, notification, and asynchronous operations. The functional API packages on the other side comprise those objects and methods, which provide access to remote resources and entities, and are thus the focus of the grid aspect of the SAGA semantic.

Error handling in SAGA is exception based. For language bindings which do not offer exceptions, SAGA requires the implementation of an error_handler interface, which allows the application to push for exceptions. No middleware exceptions are allowed to surface on application level, so all of these exceptions are mapped to the suitable SAGA exception type.

URLs are used throughout SAGA to identify different types of remote entities, such as job services, files, stream endpoints, or RPC services. In the SAGA API, binary opaque data is handled, for example on file I/O, stream I/O, or are RPC in/out parameters. The io_buffer class in the SAGA Look-&-Feel provides a common syntax and semantics, such as memory management and data store/retrieve functionality, for the various SAGA buffer types.

All SAGA object instances live within a SAGA session, which is identified by an associated set of SAGA contexts. Multiple sessions can co-exist in an application, but should not influence each other. A SAGA context represents an external SAGA security token. Remote interactions on objects which live in a specific session are bound to use one of the security tokens pointed to by the contexts attached to that session, and no other. There is no single security model, but there are interfaces to various security models.

Many SAGA objects implement the permission model, which allows to specify and control permissions on individual object methods. For example, the inquiry about a job state requires Query permissions on that job. Not many Grid back ends support that fine grained permissions, but at least for the file and replica packages, that model maps very well to the known Unix file system permission model.

SAGA attributes are basically simple key-value pairs which are attached to SAGA object class instances. Those classes need then to implement the SAGA attribute interface, which allows to set, to get and to inspect attributes.

Asynchronous notifications are a very useful paradigm for distributed programming. SAGA supports notifications on well defined metrics. The application programmer can register a custom callback for a metric, which gets invoked whenever that metric's value changes.

The SAGA task model specifies that for all classes which implement it, each method call comes in different 'flavors': synchronous, asynchronous and Task. The asynchronous and task versions return a task object instance which acts as a handle to the asynchronous operation that task represents. Tasks are stateful, and collections of tasks and jobs can be managed by task containers.

The SAGA API is extensible: the Look-&-Feel is considered to be rather stable and invariant over the API packages, but it is fairly straight forward to define additional API packages, covering new aspects of grid programming.

The functional packages defined are: job, namespace, file, replica, stream and rpc.

Starting and managing remote application instances is, along with remote data management and access, the mostly used ability of grid systems. The SAGA job packages follows the most common usage pattern: a job description (which is using the job submission description language attribute names and semantics) is submitted to a job service endpoint, which returns a handle to a stateful application instance (job). The package allows to monitor job state, and other metrics, and to inquire about a variety of job runtime attributes. Also, the package allows to actively request state changes. Additionally, the SAGA job package allows to create an additional job handle, job_self, which refers to the current application instance.

Hierarchical namespaces are a fundamental concept for a variety of grid objects, such as file systems, replica services, information directories, etc. SAGA includes thus a namespace package which provides a common base interface for all of these classes. Its design is very conventional: namespace directories can contain namespace entries and other directories, and various methods like copy(), link(), find() etc. are provided to inspect and manipulate the resulting hierarchy.

Inheriting the SAGA namespace classes, the file package is representing physical directory hierarchies. It adds the ability to perform read/write operations on the individual entries (files).

Also inheriting the SAGA namespace classes, the replica package is representing logical file hierarchies. It adds the ability to manipulate the physical replica entries a logical file is associated with, and the ability to query and manage meta data (sets of key/value pairs) attached to each logical file or directory.

Appendix G

JSAGA

Besides implementing the SAGA specification, JSAGA is also using it to enable seamless job submission to existing grid infrastructures. It deals with grid infrastructures heterogeneity (e.g. network filtering rules, supported certificate authorities, commands and services available on execution sites), in order to run collections of jobs on several infrastructures efficiently and seamlessly.

The JSAGA implementation of the Job, Namespace, File and Replica functional packages, as well as the Context Look-&-Feel package, is based on adaptors, as described earlier. These adaptors support components from various grid middleware: gLite, GT (pre-WS and WS), Unicore. There are adaptors providing support for execution management service gLite-CREAM CE, for WMS execution management technology, for VOMS security mechanism, for version 2.2 of the data management protocol SRM(Storage Resource Manager). They also support more commonly used technologies, such as X509, HTTPS, SFTP, SSH, Java keystore.

To submit jobs to different middleware first the necessary security measures need to be satisfied. The security contexts are specified in *$JSAGA_HOME/etc/jsaga_universe.xml*. The JSAGA API support RFC 3820 compliant proxies, globus proxies and old (or legacy) proxies. To specify the proxy type, the mentioned xml files need to contain a *ProxyType* attribute in the security context which should be used. This attribute can take the following values : *old, globus* or *RFC3820*. A given security context can allow full delegation of a proxy, or only limited delegation or it can prohibit it. Next the supported security contexts are presented for the different type of middleware systems for job submission (table G.1) and also for different data protocols (table G.2) and afterwards the required attributes are enumerated for different security context (table G.3).

Middleware type	Security contexts
Globus	VOMS proxy with MyProxy server, VOMS proxy,Globus proxy, Globus RFC 3820 compliant proxy,Globus legacy proxy
CREAM CE	VOMS proxy with MyProxy server, VOMS proxy, Globus proxy, Globus RFC 3820 compliant proxy,Globus legacy proxy
LCG CE	VOMS proxy with MyProxy server, VOMS proxy, Globus proxy, Globus RFC 3820 compliant proxy,Globus legacy proxy
gLite WMS	VOMS proxy with MyProxy server ,

	VOMS proxy,Globus proxy, Globus RFC 3820 compliant proxy,Globus legacy proxy

Table G.1: JSAGA security contexts

Data protocol	Security contexts
gsiftp (v1, v2, dpm)	VOMS proxy with MyProxy server ,VOMS proxy, Globus proxy, Globus RFC 3820 compliant proxy, Globus legacy proxy
gsiftp (win)	Globus proxy,Globus RFC 3820 compliant proxy, Globus legacy proxy
https	X509, JKS
irods	VOMS proxy with MyProxy server ,VOMS proxy, Globus proxy, Globus RFC 3820 compliant proxy, Globus legacy proxy
srb	VOMS proxy with MyProxy server ,VOMS proxy, Globus proxy,Globus RFC 3820 compliant proxy, Globus legacy proxy

Table G.2: JSAGA security contexts for data protocols

With the *jsaga-help.sh -s missing* those attributes can be seen which are missing from the *$JSAGA_HOME/etc/jsaga-universe.xml*. These attributes are grouped according to the available security contexts.

To specify a security context a few attributes need to be set. (table G.3)

Security context type	Security context attributes
Globus	**UserCertKey OR (UserCert AND UserKey)** - if not set, the API tries to find it interrogating the X509_USER_CERT and X509_USER_KEY environmental variables and it also looks in the HOME/.globus directory **UserProxy** - if not set, the API tries to find it searching in the /tmp directory **LifeTime** - set by default to 12 hours **Delegation** - set by default to full, it is specified optionally **CertRepository** - if not set, the API tries to find it interrogating the X509_CERT_DIR environmental variable and it also looks in the HOME/.globus/certificates directory
VOMS	**UserCertKey OR (UserCert AND UserKey)** - if not set, the API tries to find it interrogating the X509_USER_CERT and X509_USER_KEY environmental variables and it also looks in the HOME/.globus directory **UserProxy** - if not set, the API tries to find it searching in the /tmp directory **Server** - specify the voms server **UserVO** - specify the VO name the user belongs to **LifeTime** - set by default to 12 hours **Delegation** - set by default to full, it is specified optionally **ProxyType** - specify the proxy type - old, globus or RFC3820

	CertRepository - if not set, the API tries to find it interrogating the X509_CERT_DIR environmental variable and it also looks in the HOME/.globus/certificates directory **VomsDir** - where the voms server certificate is. If not specified, the API tries to find it interrogating the X509_VOMS_DIR environmental variable and it also looks in the HOME/.globus/vomsdir directory and in /etc/grid-security/vomsdir
GlobusLegacy	**UserCertKey OR (UserCert AND UserKey)** - if not set, the API tries to find it interrogating the X509_USER_CERT and X509_USER_KEY environmental variables and it also looks in the HOME/.globus directory **UserProxy** - if not set, the API tries to find it searching in the /tmp directory **LifeTime** - set by default to 12 hours **Delegation** - set by default to full, it is specified optionally **CertRepository** - if not set, the API tries to find it interrogating the X509_CERT_DIR environmental variable and it also looks in the HOME/.globus/certificates directory
GlobusRFC3820	**UserCertKey OR (UserCert AND UserKey)** - if not set, the API tries to find it interrogating the X509_USER_CERT and X509_USER_KEY environmental variables and it also looks in the HOME/.globus directory **UserProxy** - if not set, the API tries to find it searching in the /tmp directory **LifeTime** - set by default to 12 hours **Delegation** - set by default to full, it is specified optionally **CertRepository** - if not set, the API tries to find it interrogating the X509_CERT_DIR environmental variable and it also looks in the HOME/.globus/certificates directory
VOMS MyProxy	**UserCertKey OR (UserCert AND UserKey)** - if not set, the API tries to find it interrogating the X509_USER_CERT and X509_USER_KEY environmental variables and it also looks in the HOME/.globus directory **UserProxy** - if not set, the API tries to find it searching in the /tmp directory **Server** - specify the voms server **UserVO** - specify the VO name the user belongs to **LifeTime** - set by default to 12 hours **Delegation** -set by default to full,it is specified optionally **ProxyType** - specify the proxy type -old, globus or RFC3820 **DelegationLifeTime** - specify the delegation lifetime - by default it is 12 hours **CertRepository** - if not set, the API tries to find it interrogating the X509_CERT_DIR environmental variable and it also looks in the HOME/.globus/certificates directory **MyProxyServer** -specify the MyProxy server address **MyProxyUserID** -specify the User ID on the MyProxy server **MyProxyPass** - specify the password of the

	MyProxy server account

Table G.3: JSAGA security context attributes

For data management the following commands are useful: *saga-cp.sh*, *jsaga-mkdir.sh*, *jsaga-cat.sh*, *jsaga-chmod.sh*, *jsaga-ls.sh*, *jsaga-rm.sh* and *jsaga-rmdir.sh*.

Submitting jobs is done with the *jsaga-job-run.sh* command. Job status can be obtained with *jsaga-job-status.sh* and in case of some adaptor (e.g. WMS) even more information can be obtained with the *jsaga-job-info.sh*. The job output can be obtained with *jsaga-job-get-output.sh*.

All the above mentioned commands have a programmatic equivalent.

To install JSAGA the following software is required: JDK 1.5 (or above).

G.1 Transferring files using SRM

In case the SRM adaptor is installed, in the *$JSAGA_HOME/etc/jsaga-universe.xml* it can be easily specified that the srm file adaptor has to be used. The only attribute that has to be added to a context is the following: <data **type**="srm"/> After this the jsaga−mkdir.sh, jsaga−cp.sh, jsaga−ls.sh and jsaga−rm.sh commands can be used:

```
tunde@schrift:~/SAGA/jsaga_srm$ jsaga-mkdir.sh
 srm://tbn18.nikhef.nl:8446/dpm/nikhef.nl/home/pvier/tbalint
tunde@schrift:~/SAGA/jsaga_srm$ jsaga-cp.sh
 file:///home/tunde/SAGA/setUp_globus.sh
 srm://tbn18.nikhef.nl:8446/dpm/nikhef.nl/
 /home/pvier/tbalint/test.txt
tunde@schrift:~/SAGA/jsaga_srm$ jsaga-ls.sh
 srm://tbn18.nikhef.nl:8446/dpm/nikhef.nl/home/pvier/tbalint
test.txt
tunde@schrift:~/SAGA/jsaga_srm$ jsaga-rm.sh
 srm://tbn18.nikhef.nl:8446/dpm/nikhef.nl/
 /home/pvier/tbalint/test.txt
```

G.2 Running a Globus job on LCG CE

First of all a security context is needed. Let's assume, that we have a globus security context (just to be able to exemplify several security contexts). This means that the $JSAGA_HOME/etc/jsaga-universe.xml file should contain the following lines:

```
<GRID name="lcgce" contextType="Globus">
 <attribute name="UserCert"
 value="/home/tunde/.globus/usercert.pem"/>
 <attribute name="UserKey"
 value="/home/tunde/.globus/userkey.pem"/>
 <attribute name="UserProxy" value="/tmp/x509up_u1000"/>
 <attribute name="Delegation" value="full"/>
 <attribute name="CertRepository"
 value="/home/tunde/.globus/certificates/"/>
 <job type="lcgce"/>
 <data type="gsiftp-v2"/>
</GRID>
```

If the data protocol isn't specified correctly (suppose it only says 'gridftp') and the SRM adaptor isn't installed, then trying to obtain a file results in an ambiguity error.

```
tunde@schrift:~$ jsaga-cat.sh
  gsiftp://tbn14.nikhef.nl/tmp/output.txt file:$HOME/out.txt
Exception in thread "main" Ambiguity: [gsiftp] several data
  services match hostname creamce.gina.sara.nl
```

If the previously mentioned attribute isn't specified, a workaround would be to specify the gridftp version in the command line:

```
tunde@schrift:~$ jsaga-cat.sh
  gsiftp-v2://tbn14.nikhef.nl/tmp/output.txt file:$HOME/out.txt
```

Enabling the security context:

```
tunde@schrift:~/SAGA/JSAGA$ jsaga-context-init.sh lcgce
Enter UserPass for security context: lcgce
Your identity: O=dutchgrid,O=users,O=nikhef,CN=Tunde Balint
Creating proxy, please wait...
Your proxy is valid until Wed Nov 18 00:02:25 CET 2009
```

Submitting a job to the Job Manager:

```
tunde@schrift:~/SAGA/JSAGA$ jsaga-job-run.sh -Output output.txt
  -WorkingDirectory /tmp -b -Executable /bin/date -r
  gatekeeper://tbn14.nikhef.nl:2119
[gatekeeper://tbn14.nikhef.nl:2119]-
  [https://tbn14.nikhef.nl:20001/31284/1258471581/]
```

Checking the status:

```
tunde@schrift:~/SAGA/JSAGA$ jsaga-job-status.sh
  [gatekeeper://tbn14.nikhef.nl:2119]-
  [https://tbn14.nikhef.nl:20001/31284/1258471581/]
Job done.
```

Obtaining the output can be done using the *jsaga-cp.sh* command, by copying the file to which the standard output was redirected. In the same way the standard error can be redirected too.

```
tunde@schrift:~/SAGA/JSAGA$ jsaga-cp.sh
  gsiftp://tbn14.nikhef.nl/tmp/output.txt file://$HOME/out.txt
```

Trying to redirect the standard input, will result in the following RSL file.

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -Output
  output.txt -WorkingDirectory /tmp -d -Input input.txt -b
  -Executable /bin/cat -Arguments input.txt -r
  gatekeeper://tbn14.nikhef.nl:2119
&
(executable = /bin/cat)
(arguments = input.txt)
(two_phase=300)
(stdout = output.txt)
(stderr = stderr.txt)
(stdin = input.txt)
(directory = /tmp)
```

Submitting this file will fail, since the file input.txt isn't staged. To stage the file the *-FileTransfer* option should be used, but trying to specify this, won't work correctly, since the file staging isn't specified in the RSL file:

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -d
 -Output output.txt -WorkingDirectory /tmp -b -Executable
 /bin/cat -Arguments input.txt -Input input.txt -FileTransfer
 'input.txt\>input.txt' -r gatekeeper://tbn14.nikhef.nl:2119
[2009-12-08 15:06:07,043] INFO
 fr.in2p3.jsaga.helpers.xslt.XSLLogger :
Ignoring attribute 'Input' because job is interactive
[2009-12-08 15:06:07,044] INFO
 fr.in2p3.jsaga.helpers.xslt.XSLLogger :
Ignoring attribute 'Output' because job is interactive
&
(executable = /bin/sh)
(two_phase=300)
(stdout = stdout.txt)
(stderr = stderr.txt)
(directory = /tmp)
```

So specifying the FileTransfer and the Input options too, will result in a warning that the job is in Interactive mode.

Unfortunately the LCG-CE adaptor isn't developed anymore, so the encountered problems weren't rectified. A work-around to these problems is to submit a job to the WMS and specify a queue:

In a Java program the Queue can be specified in the JobDescription in the following way:

```
desc.setAttribute(JobDescription.QUEUE,
 "trekker.nikhef.nl:2119/jobmanager-pbs-short");
```

In command line this can be done the following way (trekker.nikhef.nl is an LCG-CE):

```
$jsaga-job-run.sh  -b -Executable /bin/hostname
 -WorkingDirectory /tmp -Output out.txt -Queue
 trekker.nikhef.nl:2119/jobmanager-pbs-short -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
```

Then the output can be obtained with jsaga-job-get-output.sh. The drawback is that to be able to submit this way we need a VOMS proxy. The retrieved output contains the name of the worker node on which the job was run. The job was submitted to the specified CE queue, which distributed it.

G.3 Running a gLite WMS job

In order to set run jobs on gLite, let's set up a VOMS security context, the $JSAGA_HOME/etc/jsaga-universe.xml should contain the following lines:

```
<GRID name="glite" contextType="VOMS">
  <attribute name="USERCERT"
 value="/user/tbalint/.globus/usercert.pem"/>
  <attribute name="USERKEY"
 value="/user/tbalint/.globus/userkey.pem"/>
```

```
  <attribute name="UserVO" value="pvier"/>
  <attribute name="Server"
 value="voms://voms.grid.sara.nl:30000/O=dutchgrid
 /O=hosts/OU=sara.nl/CN=voms.grid.sara.nl"/>
  <attribute name="VomsDir" value="etc/vomsdir"/>
  <job type="wms">
    <attribute name="RetryCount" value="0"/>
    <attribute name="OutputStorage" value="/tmp/sandbox"/>
  </job>
  <data type="gsiftp-v2"/>
</GRID>
```

If the data protocol isn't specified correctly (suppose it only says 'gridftp') and the SRM adaptor isn't installed, then trying to obtain a file results in an ambiguity error. Enabling the security context:

```
tunde@schrift:~/SAGA/JSAGA/jsaga-0.9.9-SNAPSHOT$
 jsaga-context-init.sh glite
Enter UserPass for security context: glite
```

Submitting a job to the WMS:

```
tunde@schrift:~/SAGA/JSAGA/jsaga-0.9.9-SNAPSHOT$
 jsaga-job-run.sh -WorkingDirectory /tmp -Output output.txt
 -Error error.txt -b -Executable /bin/date -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
[wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server] -
 [https://grasveld.nikhef.nl:9000/rT6lqXcTrYG7kHz5pmyHTw]
```

Checking the status:

```
tunde@schrift:~/SAGA/JSAGA/jsaga-0.9.9-SNAPSHOT$
 jsaga-job-status.sh
 [wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server] -
 [https://grasveld.nikhef.nl:9000/rT6lqXcTrYG7kHz5pmyHTw]
Job done.
```

Obtaining the output:

```
tunde@schrift:~/SAGA/JSAGA/jsaga-0.9.9-SNAPSHOT$
 jsaga-job-get-output_WMS.sh
 [wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server]
 -[https://grasveld.nikhef.nl:9000/rT6lqXcTrYG7kHz5pmyHTw]
Job output have been retrieved successfully
```

Redirecting the standard input can be done using the *-Input* option of the *jsaga-job-run.sh* command.

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -d
 -WorkingDirectory /tmp -Output output.txt -Error error.txt -b
 -Executable /bin/cat -Input
 /home/tunde/SAGA/jsaga_creamDebug/input.txt -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
Type = "Job";
Executable = "/bin/cat";
Requirements = true
```

```
&& (other.GlueCEStateStatus=="Production");
Rank = (-other.GlueCEStateEstimatedResponseTime);
StdInput = "/home/tunde/SAGA/jsaga_creamDebug/input.txt";
StdOutput = "output.txt";
StdError = "error.txt";
OutputSandbox = {"output.txt", "error.txt"};
OutputSandboxDestURI = {"output.txt", "error.txt"};
```

This sets the StdInput value, but the InputSandbox isn't specified. Such a job will not run on a WMS, but still, when running this job description with JSAGA, it will still run 'successfully', without reporting any errors, it will even state that it retrieved the output successfully, but since there is nothing to retrieve, we don't really know what went wrong.

The InputSandbox and/or the OutputSandbox can be specified using the *-FileTransfer* option, and then specifying *input >input* and/or *output <output* and/or *input >>input* and/or *output <<output*. Running the *jsaga-run-job.sh* command from bash this sequence has to be escaped so the standard input/output of the command isn't redirected.

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -d -b
 -WorkingDirectory /tmp -Output output.txt -Error error.txt
 -Input /home/tunde/SAGA/jsaga_creamDebug/input.txt
 -FileTransfer 'input.txt\>input.txt' -Executable /bin/cat -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
Type = "Job";
Executable = "/bin/cat";
Arguments = " input.txt";
Requirements = true
&& (other.GlueCEStateStatus=="Production");
Rank = (-other.GlueCEStateEstimatedResponseTime);
StdInput = "/home/tunde/SAGA/jsaga_creamDebug/input.txt";
StdOutput = "output.txt";
StdError = "error.txt";
InputSandbox =
{"/home/tunde/SAGA/jsaga_creamDebug/input.txt", "input.txt"};
OutputSandbox = {"output.txt", "error.txt"};
OutputSandboxDestURI = {"output.txt", "error.txt"};
OutputStorage = "/tmp";
```

But this job description will result in a error:

```
Exception in thread "main" NoSuccess:
  org.glite.wms.wmproxy.ServiceException
        at
 fr.in2p3.jsaga.adaptor.wms.job.WMSJobControlAdaptor.submit
        (WMSJobControlAdaptor.java:344)
        at
 fr.in2p3.jsaga.impl.job.instance.AbstractSyncJobImpl.doSubmit
        (AbstractSyncJobImpl.java:188)
        at fr.in2p3.jsaga.impl.task.AbstractTaskImpl.run
        (AbstractTaskImpl.java:101)
        at
 fr.in2p3.jsaga.impl.job.instance.JobImpl.run(JobImpl.java:43)
        at fr.in2p3.jsaga.command.JobRun.main(JobRun.java:117)
```

```
Caused by: org.glite.wms.wmproxy.ServiceException
        at org.glite.wms.wmproxy.WMProxyAPI.jobRegister
        (WMProxyAPI.java:530)
        at
 fr.in2p3.jsaga.adaptor.wms.job.WMSJobControlAdaptor.submit
        (WMSJobControlAdaptor.java:292)
```

This can be explained by trying to submit the same job description directly, not using the JSAGA API. Then the error received explains why this JDL file isn't correct:

```
InputSandbox: filename conflict found while extracting files.
The following file is repeated more than once: 'input.txt'
```

The job description which can be successfully submitted and the output can be retrieved is:

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -d -b
 -WorkingDirectory /tmp -Output output.txt -Error error.txt
 -FileTransfer 'input.txt\>input.txt' -Executable /bin/cat
 -Arguments input.txt -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
Type = "Job";
Executable = "/bin/cat";
Arguments = " input.txt";
Requirements = true
&& (other.GlueCEStateStatus=="Production");
Rank = (-other.GlueCEStateEstimatedResponseTime);
StdOutput = "output.txt";
StdError = "error.txt";
InputSandbox = {"input.txt"};
OutputSandbox = {"output.txt", "error.txt"};
OutputSandboxDestURI = {"output.txt", "error.txt"};
OutputStorage = "/tmp";
```

So the standard input isn't redirected, but the InputSandbox is specified and staging of the input files is done. Another solution would be to specify the standard input, but in this case another file has to be specified in the InputSandbox. If nothing is specified in the InputSandbox, then the file specified as the standard input file isn't staged. When specifying a list of arguments, the argument list has to be escaped (even the space):

```
tunde@schrift:~/SAGA/jsaga_creamDebug$ jsaga-job-run.sh -d -b
 -WorkingDirectory /tmp -Input input.txt -Output output.txt
 -FileTransfer 'input1.txt\>input1.txt' -Error error.txt
 -Executable /bin/cat -Arguments 'input.txt\ input1.txt' -r
 wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server
Type = "Job";
Executable = "/bin/cat";
Arguments = " input.txt input1.txt";
Requirements = true
&& (other.GlueCEStateStatus=="Production");
Rank = (-other.GlueCEStateEstimatedResponseTime);
StdInput = "input.txt";
StdOutput = "output.txt";
StdError = "error.txt";
```

```
InputSandbox = {"input.txt", "input1.txt"};
OutputSandbox = {"output.txt", "error.txt"};
OutputSandboxDestURI = {"output.txt", "error.txt"};
OutputStorage = "/tmp";
```

G.4 Running a job on CREAM CE

While trying to submit a cream job, a few not so common problems can occur. First it may be that JSAGA starts complaining about the HTTPS protocol.

```
tunde@schrift:~/SAGA/JSAGA/jsaga-0.9.9-SNAPSHOT$
 jsaga-job-run.sh -Output output.txt -WorkingDirectory /tmp -b
 -Executable /bin/date -r
 cream://creamce.sara.gina.nl/cream-pbs-short
Exception in thread "main" AuthenticationFailed: No client
 transport named 'https' found!
```

The solution for this problem is to reinstall JSAGA without the SRM plug-in. With the JSAGA cream adaptor the HTTPS transport is used to contact the delegation service.

Globus supports a GSI SSL enabled HTTP protocol called HTTPG. GT 3.x Java WS Core security provided support for web services to use HTTPG to secure the server-client interaction and for delegation of client credentials to the server, but with GT 4.x, the HTTPG use in Java WS Core was deprecated and discontinued. Instead Globus advocates the use of HTTPS in conjunction with a Delegation Service for delegation of client credentials. In this method, standard HTTPS (or HTTP with other forms of security) can be used for authentication and securing the application data, and delegation of credential is done outside of the handshake protocol. The Delegation Service provides a remote endpoint for the client to contact and delegate its credentials, and returns an identifier to the delegated credential. The client can then provide the identifier, as a part of the application protocol, to any entity it wants to delegate to. The Delegation Service can be set up with access policy per delegated credential to ensured restricted and secured access.

A second encountered problem involves the proxy lifetime. The signer certificate expires earlier than the delegated proxy if the VOMS lifetime is lower than 12H. This can be fixed by setting the LifeTime attribute in the *jsaga-universe.xml* file. This means that in order to obtain a valid security context the following lines need to be present in the *jsaga-univers.xml* file:

```
<GRID name="glite" contextType="VOMS">
  <attribute name="USERCERT"
 value="/home/tunde/.globus/usercert.pem"/>
  <attribute name="USERKEY"
 value="/home/tunde/.globus/userkey.pem"/>
  <attribute name="UserProxy" value="/tmp/x509up_u1000"/>
  <attribute name="Server"
 value="voms://voms.grid.sara.nl:30000/O=dutchgrid/
O=hosts/OU=sara.nl/CN=voms.grid.sara.nl"/>
  <attribute name="UserVO" value="pvier"/>
  <attribute name="LifeTime" value="PT13H"/>
  <attribute name="Delegation" value="full"/>
  <attribute name="ProxyType" value="old"/> <!-- old globus or
 RFC3820 -->
```

```
  <attribute name="CertRepository"
  value="/home/tunde/.globus/certificates/"/>
  <attribute name="VomsDir" value="/home/tunde/.globus"/>
  <job type="cream"/>
  <data type="gsiftp-v2"/>
</GRID>
```

If also job to the WMS need to be submitted, we can have several job attribute types, without repeating the whole security context. If the security context is repeated an ambiguity error will prevent us from submitting jobs, since the API will not distinguish successfully which context to use.

After this the context can be initialized, and jobs can be submitted to the CREAM CE. The staging of files is done using GridFTP. To successfully submit a job a JDL file is required. This file can be automatically generated by JSAGA. A problem occurs when we don't have the JDL file which should be submitted. In case the JDL file is generated using the *'-d'* option of *jsaga-job-run.sh* command, and then submitted using the *'–file'* option, then the output is returned, otherwise JSAGA generates a unique ID which is used to specify where to copy the output files after the job is finished, but this unique ID is lost (cannot be obtained anymore) and this way the user doesn't know from which gridftp server to retrieve the output. And there is no command implemented yet in JSAGA, which would actually retrieve the specified files. Initialize security context:

```
tunde@schrift:~/SAGA/jsaga_noSRM$ jsaga-context-init.sh glite
Enter UserPass for security context: glite
```

Generating a JDL file without redirecting the standard output, submitting the job and not being able to retrieve output:

```
tunde@schrift:~/SAGA/jsaga_noSRM$ jsaga-job-run.sh -d -b
 -Executable /bin/date -r
 cream://creamce.gina.sara.nl:8443/cream-pbs-short > submit.jdl
tunde@schrift:~/SAGA/jsaga_noSRM$ cat submit.jdl
[
  Type = "Job";
  BatchSystem    = "pbs";
  QueueName      = "short";
  Executable = "/bin/date";
  Requirements = true ;
  Rank = -other.GlueCEStateEstimatedResponseTime ;
  RetryCount = 0;
]
tunde@schrift:~/SAGA/jsaga_noSRM$ jsaga-job-run.sh --file
 ./rex.txt -b -Executable /bin/date -r
 cream://creamce.gina.sara.nl:8443/cream-pbs-short
[cream://creamce.gina.sara.nl:8443/cream-pbs-short] -
 [CREAM719929565]
```

Output file cannot be obtained.

```
tunde@schrift:~/SAGA/jsaga_noSRM$ jsaga-job-run.sh -d -Output
 output.txt -WorkingDirectory /tmp -b -Executable /bin/date -r
 cream://creamce.gina.sara.nl:8443/cream-pbs-short > submit.jdl
tunde@schrift:~/SAGA/jsaga_noSRM$ cat submit.jdl
[
```

```
  Type = "Job";
  BatchSystem    = "pbs";
  QueueName      = "short";
  Executable = "/bin/date";
  StdOutput = "output.txt";
  OutputSandbox = {"output.txt"};
  OutputSandboxBaseDestURI =
      "gsiftp://creamce.gina.sara.nl/tmp/1259751956485/";
  Requirements = true ;
  Rank = -other.GlueCEStateEstimatedResponseTime ;
  RetryCount = 0;
]
tunde@schrift:~/SAGA/jsaga_noSRM$ jsaga-job-run.sh --file
 ./submit.jdl -b -Executable /bin/date -r
 cream://creamce.gina.sara.nl:8443/cream-pbs-short
[cream://creamce.gina.sara.nl:8443/cream-pbs-short] -
 [CREAM904598737]
```

Obtain the job status:

```
tunde@schrift:~$ jsaga-job-status.sh
 [cream://creamce.gina.sara.nl:8443/cream-pbs-short] -
 [CREAM467587458]
Job done.
```

Output file can be obtained. Also transferring arguments for the executable is successful.

Another observation is that specifying the JDL file with the *-f* or *–file* options, doesn't mean that the JDL file is parsed. Since we *need to* specify the Executable too, this is taken into account and the output isn't redirected, if not specified explicitly, although the JDL file contains attributes which would redirect the output. According to the developers, this is the expected behavior: the job description file must only contain SAGA job description attributes (the idea is to enable to submit the same job description to several middleware), and these attributes can be overwritten by command line attributes for convenience. Thus the option -file is for SAGA attributes only.

If the data protocol isn't specified correctly in the *$JSAGA_HOME/etc/jsaga-univers.xml* file (suppose it only says 'gridftp') and the SRM adaptor isn't installed, then trying to obtain a file results in an ambiguity error.

```
tunde@schrift:~$ jsaga-cat.sh
 gsiftp://creamce.gina.sara.nl/tmp/output.txt
Exception in thread "main" Ambiguity:
  [gsiftp] several data services match
  hostname creamce.gina.sara.nl
```

If the previously mentioned attribute isn't specified, a workaround would be to specify the gridftp version in the command line:

```
tunde@schrift:~$ jsaga-cat.sh
 gsiftp-v2://tbn14.nikhef.nl/tmp/output.txt file:$HOME/out.txt
```

If the job is submitted using the interactive mode, meaning that the command won't exit immediately after submitting the job and returning the job ID, then we will get the expected result, although we'll have to wait until the job finishes.

```
tunde@schrift:~$ jsaga-job-run.sh -Executable /bin/date -r
 cream://creamce.gina.sara.nl:8443/cream-pbs-short > rez.txt
```

```
tunde@schrift:~$ cat rez.txt
Wed Dec  2 11:06:43 CET 2009
```

G.5 Programming using JSAGA

Unfortunately when writing a Java program which uses the JSAGA libraries, we'll run into the most of the issues, which were mentioned until now. A short test program *RunJobJSAGA.java* was written, which tries to create the necessary security context, and then tries to run a job. The program takes as a command line argument the following options:

1. *-context* - to specify which type of context has to be initialized - e.g voms, globus,....
2. *-server* - to specify where to job should be run.
   E.g cream://creamce.gina.sara.nl:8443/cream-pbs-short
   OR gatekeeper://tbn14.nikhef.nl:2119/jobmanager
   OR wms://graszode.nikhef.nl:7443/glite_wms_wmproxy_server

Creating a security context:

```
Context context;
context = ContextFactory.createContext(contextType);
context.setAttribute(Context.SERVER,
 "voms://voms.grid.sara.nl:30000/" +
 "O=dutchgrid/O=hosts/OU=sara.nl" + "/CN=voms.grid.sara.nl");
String user_home = System.getenv("HOME");
// Define your VO
context.setAttribute(Context.USERVO, "pvier");
// The path of your user certificate
context.setAttribute(Context.USERCERT, user_home +
 "/.globus/usercert.pem");
// The path of your user key
context.setAttribute(Context.USERKEY, user_home +
 "/.globus/userkey.pem");
// A directory containing the CA certificates
context.setAttribute(Context.CERTREPOSITORY, user_home +
 "/.globus/certificates");
context.setAttribute(VOMSContext.VOMSDIR, "etc/vomsdir");
context.setAttribute(Context.LIFETIME, "PT13H");
context.setDefaults();
System.out.println("Enter UserPass for security context: " +
 context.getAttribute(Context.TYPE));
String userPass = getPassword();
// set UserPass
if (userPass != null) {
  context.setAttribute(Context.USERPASS, userPass);
}
```

The previous attributes depend on the security context, so some of them can be deleted for some contexts. (See the table which specifies which attributes have to be set for which security context G.3). After this a session can be created and the security context can be added to it:

```
Session session = SessionFactory.createSession(false);
session.addContext(context);
```

Creating the server URL:

```
URL url = URLFactory.createURL(server);
```

Creating the job description:

```
JobDescription desc = JobFactory.createJobDescription();
desc.setAttribute(JobDescription.EXECUTABLE, "/bin/cat");
desc.setVectorAttribute(JobDescription.ARGUMENTS, new String[]
 { "input.txt","input1.txt" });
desc.setAttribute(JobDescription.WORKINGDIRECTORY, workingDir);
desc.setAttribute(JobDescription.INPUT, "input.txt");
desc.setAttribute(JobDescription.OUTPUT, "out1.txt");
desc.setAttribute(JobDescription.ERROR, "err.txt");
if (url.getScheme().equalsIgnoreCase("wms")) {
  desc.setAttribute(JobDescription.INPUT, "input.txt");
  desc.setVectorAttribute(JobDescription.FILETRANSFER, new
 String[] {  "input1.txt"+">"+"input1.txt", "out.txt" + "<" +
 workingDir + "out1.txt", "err.txt" + "<" + workingDir +
 "err.txt"});
  }
desc.setAttribute("OutputStorage", "/tmp");
```

When running the job on the WMS, input files can be transfered successfully, and the output can be retrieved just by specifying it as *JobDescription.FILETRANSFER*. The output in our case will be in /tmp/out1.txt.

After creating the job description, a job service needs to be created and then this can be used to create, submit the job and get its state.

```
JobService service = JobFactory.createJobService(session, url);
Job job = service.createJob(desc);
job.run();
job.waitFor();
State s = job.getState();
```

When running a job on the LCG CE, an URL had to be created to be able to retrieve the output, but this can be deduced from the job description. This is done the following way (create URL and display content of file):

```
if (s==State.DONE){
  if (url.getScheme().equalsIgnoreCase("gatekeeper"))
  {
    //retrieve output and error files
    URL url_file =
 URLFactory.createURL("gsiftp://"+url.getHost() +
 workingDir+desc.getAttribute(JobDescription.OUTPUT));
    File f = FileFactory.createFile(session, url_file);
    long  fSize = f.getSize();
    System.out.println(fSize);
    Buffer buffer = BufferFactory.createBuffer(new byte[1024]);
    f.read(buffer);
    byte[] bufD = buffer.getData();
```

```
      System.out.println(new String(bufD));
      System.exit(0);
    }
}
```

If we try to set in the job description *JobDescription.INPUT* and run the job on LCG CE, the state of the job will be: failed.

When running the program a warning was displayed. This was suppressed by setting the following:

```
//to eliminate : [main] ERROR util.FileEndingIterator
//       - Error while reading directory null
org.apache.log4j.Logger.getLogger
   (org.glite.security.util.FileEndingIterator
   .class.getName()).setLevel(org.apache.log4j.Level.FATAL);
```

Error when trying to transfer input files to a different location:

```
desc.setVectorAttribute(JobDescription.FILETRANSFER, new
 String[] { "input.txt" +">"+ workingDir + "/input.txt",
 "out.txt" + "<" + workingDir + "out1.txt", "err.txt" + "<" +
 workingDir + "err.txt"});
NoSuccess: javax.xml.transform.TransformerException:
Renaming file is not supported: input.txt / /tmp//input.txt
```

The following program sets up the necessary security context and copies a file. It takes as input arguments the source and destination, which are read in the variables *source* and *dest*.

```
context = ContextFactory.createContext(contextType);
context.setAttribute(Context.SERVER,
"voms://voms.grid.sara.nl:30000/O=dutchgrid/O=hosts/OU=sara.nl"
 + "/CN=voms.grid.sara.nl");
String user_home = System.getenv("HOME");
// Define your VO
context.setAttribute(Context.USERVO, "pvier");
// The path of your user certificate
context.setAttribute(Context.USERCERT, user_home +
 "/.globus/usercert.pem");
// The path of your user key
context.setAttribute(Context.USERKEY, user_home +
 "/.globus/userkey.pem");
// A directory containing the CA certificates
context.setAttribute(Context.CERTREPOSITORY,
 "/etc/grid-security/certificates");
context.setAttribute(VOMSContext.VOMSDIR, "etc/vomsdir");
context.setDefaults();
String userPass = getPassphrase();
// set UserPass
if (userPass != null) {
  context.setAttribute(Context.USERPASS, userPass);
}
context.getAttribute(Context.USERID);
Session session = SessionFactory.createSession(false);
```

```
session.addContext(context);
System.out.println(context.getAttribute(Context.TYPE) + "
  initialized");
URL url_source = URLFactory.createURL(source);
URL url_dest = URLFactory.createURL(dest);
File fS = FileFactory.createFile(session, url_source);
fS.copy(url_dest);
```

Using the SRM adaptor files were transfered to tbn18.nikhef.nl SRM and using the GridFTP adaptor files were transferred to hooizolder.nikhef.nl.

Errors

- Using existing proxy file
  If we have a valid proxy, which should be reused, and there is no user certificate and user key on the computer where we want to start a job with JSAGA, then setting only the USERPROXY attribute in the context in the following way:

  ```
  context.setAttribute(Context.USERPROXY, Util.proxyF);
  ```

  isn't enough. In this case it used the default proxy certificate all the time, which in my case was /tmp/x509up_u1000. This wasn't good, because I had the globus proxy certificate in this file and the proxy certificate with the VOMS attributes was saved somewhere else. This resulted in the fact that I couldn't authenticate, which is obvious, because in the certificate which was used by JSAGA I didn't have the necessary attributes. In this case JSAGA wasn't using a jsaga-universe.xml configuration file, it was trying to get the needed credentials from the default locations. I solved this by generating a jsaga-universe.xml configuration file into the /tmp directory, which basically only contained the same information that I tried to set for the Context. This looks like:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <UNIVERSE xmlns="http://www.in2p3.fr/jsaga" name="World">
    <GRID contextType="Globus" name="globus">
      <attribute name="UserProxy" value="/tmp/proxyF"/>
      <attribute name="Server"
    value="voms://voms.grid.sara.nl:30000
        /O=dutchgrid/O=hosts/OU=sara.nl/CN=voms.grid.sara.nl"/>
      <attribute name="UserVO" value="pvier"/>
    </GRID>
  </UNIVERSE>
  ```

- FileEndingIterator error
  If there is a error similar to this: org.glite.security.util.FileEndingIterator – (FileEndingIterator.java:109) : Error **while** reading directory **null** the solution is to add the following line to the code:

  ```
  org.apache.log4j.Logger.getLogger(
    org.glite.security.util.FileEndingIterator.class.getName())
    .setLevel (org.apache.log4j.Level.FATAL);
  ```

Appendix H

JavaSAGA

JavaSAGA uses

- the JavaGAT resource broker to submit jobs. This means that there are adaptors for Globus, GridSAM, SSH and others
- the File and Namespace adaptors use the JavaGAT File, FileInputStream and FileOutputStream packages. This means that there are adaptors for GridFTP, FTP, SSH and others
- the LogicalFile adaptor uses the SAGA file and namespace packages
- for the stream functional package there is Java Socket adaptor and a JavaGAT adaptor, which uses the JavaGAT endpoint package.
- the RPC package is implemented with a XMLRPC adaptor

JavaSAGA can be downloaded from:
`http://sourceforge.net/projects/saga/files/SAGA%20Java%20Implementation/`.

The following test were done with version 1.0.1. The prerequisites of installing JavaSAGA are Java SUN JDK version 6 or newer and Ant. The archive has to be extracted to a directory, the environmental variable *$JAVA_SAGA_LOCATION* has to be set to the directory where JavaSAGA should be installed and then by simply typing ***ant*** in the directory where the archive was extracted the program is installed.

H.1 Running programs

It is assumed that all the source files are in a *src* directory. Using Apache Ant a JAR file is created including the GAT-engine.jar. The used *build.xml* file looks like this:

```
<project name="JavaSAGA Tests" default="usage" basedir=".">
  <property environment="env"/>
    <property name="srcdir"   location="src"/>
    <property name="builddir" location="build"/>
    <property name="distdir"  location="jars"/>
    <property name="docdir"   location="docs"/>
    <property name="cpath"
 location="${env.JAVA_SAGA_LOCATION}/lib/saga-api-1.0.1.jar"/>
    <target name="usage" description="Print usage string">
```

```
   <echo message="JavaSAGA Tests"/>
   <echo message="  ant build  : build the test jarfile."/>
   <echo message="  ant clean  : to clean the tree."/>
   </target>
 <target name="check-environment">
       <condition property="ant.correct">
     <isset property="env.ANT_HOME"/>
       </condition>
       <condition property="saga.set">
     <isset property="env.JAVA_SAGA_LOCATION"/>
       </condition>
       <available file="${cpath}" property="saga.correct"/>
   </target>
   <target name="check-saga-correct" unless="saga.correct">
       <echo message="Your $JAVA_SAGA_LOCATION is not set
correctly (file not found)!"/>
       <fail/>
   </target>
   <target name="check-saga-set" unless="saga.set">
       <echo message="Your $JAVA_SAGA_LOCATION is not set!"/>
       <fail/>
   </target>
   <target name="check-ant-correct" unless="ant.correct">
       <echo message="Your $ANT_HOME is not set!"/>
       <fail/>
   </target>
   <target name="prepare">
       <mkdir dir="${distdir}"  />
       <mkdir dir="${builddir}" />
   </target>
   <echo message="${cpath}"/>
   <target name="perform-build"
       depends="prepare">
       <delete failonerror="false" file="${distdir}/test.jar"
/>
       <javac  srcdir="${srcdir}"
       destdir="${builddir}"
               classpath="${cpath}"
               includes="**/**"
               debug="true"
               deprecation="true" >
     <compilerarg value="-Xlint"/>
       </javac>
       <jar jarfile="${distdir}/test.jar">
           <fileset dir="${builddir}" >
               <include name="**/**" />
           </fileset>
       </jar>
       <delete failonerror="false" dir="${builddir}" />
   </target>
```

```
    <target name="build" depends="check-environment,
  check-ant-correct, check-saga-set,
  check-saga-correct,prepare,perform-build"
  description="Build the test jar file"/>
    <target name="clean"
        description="Clean the tree">
        <delete failonerror="false" dir="${distdir}" />
        <delete failonerror="false" dir="${builddir}" />
    </target>
</project>
```

After creating the test.jar file in the *jars* directory, making sure that the $GAT_LOCATION is set and that the adaptors can be found, the programs can be run. To make the execution of the programs and the setting of the classpath easier, a script was created:

```
tunde@schrift:~/JavaSAGA/saga-impl-1.0.1/javaSaga_tests$ cat
  runPrg.sh
#!/bin/sh
if [ -z "$JAVA_SAGA_LOCATION" ] ; then
  echo JAVA_SAGA_LOCATION variable not set, using $PWD/..
  JAVA_SAGA_LOCATION=$PWD/..
fi
SAGA_ENGINE_LOCATION=${JAVA_SAGA_LOCATION}/lib
SAGA_ADAPTOR_LOCATION=${JAVA_SAGA_LOCATION}/lib/adaptors
add_to_gat_classpath () {
  DIRLIBS=${1}/*.jar
  for i in ${DIRLIBS}
  do
    if [ "$i" != "${DIRLIBS}" ] ; then
      if [ -z "$SAGA_CLASSPATH" ] ; then
        SAGA_CLASSPATH=$i
      else
        SAGA_CLASSPATH="$i":$SAGA_CLASSPATH
      fi
    fi
  done
}

add_to_gat_classpath $SAGA_ENGINE_LOCATION
SAGA_CLASSPATH=$SAGA_CLASSPATH:$SAGA_ENGINE_LOCATION

echo ${SAGA_ADAPTOR_LOCATION}

java -cp :./jars/test.jar:$SAGA_CLASSPATH
  -Dsaga.adaptor.path=$SAGA_ADAPTOR_LOCATION
  -Dlog4j.configuration=
  file:$JAVA_SAGA_LOCATION/log4j.properties
  -Dsaga.location=$JAVA_SAGA_LOCATION $*
```

File transfer

The prerequisites for transferring files using a Globus proxy is to have a user certificate,

the private key, and to have the certificates installed. It is possible to create a proxy using *grid-proxy-init* provided with the implementation. In case of avoiding this command it is possible to create a security context and to tranfer a file using the following code sequence:

```
Session session = SessionFactory.createSession();
URL url = URLFactory.createURL(args[0]);
URL url1 = URLFactory.createURL(args[1]);
String scheme = url.getScheme();
String scheme1 = url1.getScheme();

if (("gsiftp".equals(scheme)) || ("gsiftp".equals(scheme1))) {
  // Gridftp context.
  Context context = ContextFactory.createContext("gridftp");
  context.setAttribute(Context.USERPASS, getPassphrase());
  session.addContext(context);
}
else if (("ftp".equals(scheme)) || ("ftp".equals(scheme1))) {
  // FTP context. Default is anonymous.
  session.addContext(ContextFactory.createContext("ftp"));
}
File source = FileFactory.createFile(session, url);
source.copy(url1, Flags.NONE.getValue());
```

The following executed successfully:

```
./runPrg.sh FileCopy file:////tmp/saga_test.txt
 gsiftp://tbn14.nikhef.nl//tmp
```

The available contexts are: ssh, sftp, ftp, gridftp, globus and preferences. The preferences context is extensible. In case of a VOMS proxy this context has to be used. Trying to transfer a file to SRM didn't work, since it constantly states that it cannot find the path.

```
{DoesNotExist: Target parent file tbalint does not exist
 /dpm/nikhef.nl/home/pvier/tbalint }
```

To run a job with globus a similar program needs to be written. A proxy certificate has to be generated beforehand. As it can be seen it uses a JavaGAT adaptor, so JavaSAGA is mostly based on the JavaGAT. It also creates a job description and runs the job.

```
import org.ogf.saga.context.Context;
import org.ogf.saga.context.ContextFactory;
import org.ogf.saga.job.Job;
import org.ogf.saga.job.JobDescription;
import org.ogf.saga.job.JobFactory;
import org.ogf.saga.job.JobService;
import org.ogf.saga.monitoring.Callback;
import org.ogf.saga.monitoring.Metric;
import org.ogf.saga.monitoring.Monitorable;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.url.URL;
import org.ogf.saga.url.URLFactory;

public class GlobusJob {
```

```
  public static void main(String[] args) {
    // Make sure that the SAGA engine
    // picks the javagat adaptor for JobService.
    System.setProperty("JobService.adaptor.name", "javaGAT");
    String server = null;
    if (args.length == 1) {
      server = args[0]; }
    else
      {
        System.out.println("Server URL has to be specified");
        System.exit(1);
      }
    System.out.println("submit job to: "+server);
    try {
      URL serverURL = URLFactory.createURL(server);
      Session session = SessionFactory.createSession(true);
      // Create a preferences context for JavaGAT.
      Context context =
 ContextFactory.createContext("preferences");
      // Make sure that javaGAT picks the globus adaptor
      context.setAttribute("ResourceBroker.adaptor.name",
 "globus");
      context.setAttribute("machine.node", serverURL.getHost());
      context.setAttribute("File.adaptor.name",
 "Local,GridFTP");
      session.addContext(context);
      // Create the JobService.
      JobService js = JobFactory.createJobService(serverURL);
      JobDescription jd = JobFactory.createJobDescription();
      jd.setAttribute(JobDescription.EXECUTABLE,
 "/bin/hostname");
      jd.setAttribute(JobDescription.NUMBEROFPROCESSES, "2");
      jd.setAttribute(JobDescription.OUTPUT, "hostname.out");
      jd.setAttribute(JobDescription.ERROR, "hostname.err");
      jd.setVectorAttribute(JobDescription.FILETRANSFER, new
 String[] { "hostname.out < hostname.out", "hostname.err <
 hostname.err"});
      // Create the job, run it, and wait for it.
      Job job = js.createJob(jd);
      job.run();
      job.waitFor();
    } catch (Throwable e) {
      System.out.println("Got exception " + e);
      e.printStackTrace();
      e.getCause().printStackTrace();
    }
  }
}
```

Then we can build it with ant, create a proxy certificate and run:

```
./runPrg.sh GlobusJob any://tbn14.nikhef.nl:2119
```

When trying to submit a job to gLite we have to make sure that the security context properties required by JavaGAT are set and that there is a valid VOMS proxy available. This can be achieved with adding to the previously defined context the following:

```
context.setAttribute(Context.USERPROXY, "/tmp/x509up_u1000");
context.setAttribute(Context.USERCERT,
 "/home/tunde/.globus/usercert.pem");
context.setAttribute(Context.USERKEY,
 "/home/tunde/.globus/userkey.pem");
context.setAttribute(Context.USERVO,"pvier");
context.setAttribute(Context.USERPASS, getPassphrase());
context.setAttribute(Context.SERVER,
 "voms://voms.grid.sara.nl:30000/O=dutchgrid/O=hosts"
 +"OU=sara.nl/CN=voms.grid.sara.nl");
context.setAttribute(Context.TYPE, "VOMS");
context.setAttribute(Context.CERTREPOSITORY,
 "/etc/grid-security/certificates/");
```

Unfortunately after this step it can be noticed that JavaSAGA creates a JDL file with some attributes set, but some of these attributes aren't supported by the JavaGAT gLite adaptor. These are: *save.state, sandbox.delete* and setting the working directory. I commented out these attributes in the JavaSAGA source code
(file: $JAVA_SAGA_LOCATION/
adaptors/JavaGat/src/org/ogf/saga/adaptors/javaGAT/job/SagaJob.java). I didn't get any exceptions regarding the proxy, but when trying to submit the job, it tries to create a directory for the input/output sandbox, with the following URI: *any://graspol.nikhef.nl/.JavaGAT_SANDBOX_0.3564132435164147* or something similar. And then it will give an error stating that mkdir didn't work, and the directory cannot be found. So as far as I understood it the working directory cannot be successfully specified.

Appendix I

SAGA C++

The SAGA C++ [43] implementation is a complete SAGA compliant implementation: it covers all functional and non-functional packages of the SAGA Core API specification. Backend bindings exist for local systems (Unix, MacOS, Windows), GT4 (GRAM, GridFTP, Replica Location Service), Condor, SSH, clouds (Amazon Elastic Compute Cloud), LSF and GridSAM (job submission and management).

A disadvantage is that if we want to build the Globus Toolkit adaptors, a local installation of the Globus C header and client library files is. The good thing is that the local Globus services don't have to be configured or running. The adaptors use the Globus GridFTP, GRAM2, RLS and their dependent client libraries. So either the developer packages for Globus need to be installed or Globus can be installed from source in order to make the Globus header files available.

In order to provide the default shallow copying of SAGA objects, meaning that no object state is copied, and to assure that the lifetime of a SAGA object is not only defined by its scope in the program, the SAGA C++ implementation uses a technique called Private Implementation(PIMPL) mechanism. The lifetime of a SAGA object [36] should depend on the lifetime of objects depending on the instance, on the pending operations for the instance and the shallow copies of the instance.

Using the PIMPL mechanism, the SAGA object does not maintain any state itself, but is merely a facade maintaining a private, shared pointer to the implementation of the (stateful) SAGA object, and all method invocations are simply forwarded to that implementation instance. On copies, a new facade instance is created which maintains another shared copy to the same implementation instance, using shallow copy semantics, as the stateful implementation is not copied at all. Also, depending objects and task instances (which represent asynchronous operations) maintain additional shared pointers to the implementationinstance and are thus extending the lifetime of that instance: only when all shared pointer copies are finally freed (i.e. when all depending objects are deleted and all asynchronous operations are completed) is the stateful implementation deleted.

There are two versions of the SAGA C++ libraries. In the standard version the adaptor libraries aren't linked to the application, they are loaded at compile time, meaning that the shared library dependencies have to be solved at runtime and SAGA has to be configured to find the adaptor libraries. In the lite version there is a single shared library which contains the SAGA engine and a set of adaptors. These adaptors are thus not loaded at runtime, but are linked at link-time, so all dependencies are resolved at link-time.

The installation is a usual configure,make,make install sequence. The only necessary

requirements are a C++ complier (gcc>=3.4) and Boost C++ Libraries(>=1.3.3). After the installation it is possible that the $LD_LIBRARY_PATH needs to be set, so the libraries can be found. Also the SAGA_LOCATION environmental variable has to point to the SAGA installation root.

To install the Globus GT4 adaptors, the environmental variable $GLOBUS_LOCATION and $GLOBUS_FLAVOR has to be set to the root directory of the Globus installation. For the Condor adaptor a working and configured Condor client is required.

If an adaptor wasn't installed, it can be added by configuring with the *–prefix=$SAGA_LOCATIONS* option. Then *make* and *make install* is required. The following environmental variables need to be set:

```
export SAGA_LOCATION=/home/tunde/SAGA/sagaC
export GLOBUS_LOCATION=/usr/local/globus-4.2.1
export LD_LIBRARY_PATH=
 /home/tunde/SAGA/sagaC/lib/:/usr/local/globus-4.2.1/lib
export PATH= $SAGA_LOCATION/bin:$GLOBUS_LOCATION/bin:$PATH
```

After installing Globus 4.2.1 from source SAGA C++ gave a Kerberos error, saying it's not able to file the kerberos cache file. This is because the SAGA C++ falled back to Kerberos authentication, instead of picking up the existing Globus proxy file, and using that for authentication. This was solved by reconfiguring SAGA C++, and allowing it to only install the Globus adaptor:

```
./configure --with-adaptor_suites=globus
 --prefix=/home/tunde/SAGA/sagaC_res/
```

Then using make and make install.

In order to see what is happening the SAGA_VERBOSE environmental variable has to be set. The values of this variable can be: 1 - critical, 2 - error, 3 - warning, 4 - info, 5 - debug, 6 - blurb.

I.1 Running programs

To be able to run a job with Globus, we need to use the *grid-proxy-init* or *voms-proxy-init* commands, to generate the proxy. There is no command in SAGA C++ which automatically generates the proxy. To see if there is a security context:

```
tunde@schrift:~/SAGA/sagaC1/adaptors/globus/job$ saga-context
 globus
 ---------------------------------
 LifeTime        : 43194
 Type    : globus
 UserID          : /O=dutchgrid/O=users/O=nikhef/CN=Tunde Balint
 UserProxy       : /tmp/x509up_u1000
 ---------------------------------
```

If no security context is initialized or if there is no adaptor which can handle the job, we get a similar error:

```
SAGA(BadParameter):
SAGA(BadParameter): default_job: Could not initialize job
 service for [gram://tbn14.nikhef.nl:2119/jobmanager]. Only
 'localhost' and schrift are supported.
```

```
SAGA(AuthorizationFailed): globus_gram_job: Could not
  initialize job service for
  [gram://tbn14.nikhef.nl:2119/jobmanager].
Credentials are invalid or do not exist (grid-proxy-init?)
SAGA(BadParameter): omii_gridsam_job: Could not initialize job
  service for [gram://tbn14.nikhef.nl:2119/jobmanager].
Only any://, gridsam:// and https:// schemes are supported.
SAGA(BadParameter): ssh_job: Cannot handle path in ssh URLs
SAGA(NoAdaptor): No adaptor succeeded in executing constructor
  for job_service_cpi
```

Setting up the security context is done in the following way from a C++ program:

```
saga::context globus_context ("globus");
globus_context.set_attribute("UserCert",
 "/home/tunde/.globus/usercert.pem");
globus_context.set_attribute("UserKey",
 "/home/tunde/.globus/userkey.pem");
globus_context.set_attribute("CertRepository",
 "/home/tunde/.globus/certificates");
globus_context.set_attribute("UserProxy", "/tmp/x509up_u1000");
```

In the command line the proxy file is detected automatically in its standard location.

Trying to manage files from command line works:

```
tunde@schrift:~/SAGA/sagaC_tests$ saga-file copy
  file://localhost/tmp/test.txt gsiftp://tbn14.nikhef.nl/tmp
tunde@schrift:~/SAGA/sagaC_tests$ saga-file cat
  gsiftp://tbn14.nikhef.nl/tmp/test.txt
this is a saga test
tunde@schrift:~/SAGA/sagaC_tests$ saga-file remove
  gsiftp://tbn14.nikhef.nl/tmp/test.txt
```

When writing a program the previously mentioned security context has to be associated to a session:

```
saga::session my_session;
my_session.add_context (globus_context);
```

To copy a file a functional SAGA call, *file.copy()*, is provided by the file class in the saga::filesystem package. This can be used in combination with saga::url to copy file between machines:

```
saga::url u("file://localhost/tmp/alma1.txt");
saga::filesystem::file f (my_session, u);
f.copy (saga::url ("gsiftp://tbn14.nikhef.nl/tmp"));
```

After this we can read the content of the copied file the following way:

```
saga::url remote("gsiftp://tbn14.nikhef.nl/tmp/alma1.txt");
saga::filesystem::file f1 (my_session, remote);
char buf[20];
saga::mutable_buffer::mutable_buffer mb(buf, 20);
f1.read(mb);
std::cout<< buf << std::endl;
```

Trying to run a job from the command line interface

```
tunde@schrift:~/SAGA/sagaC_tests$ saga-job run
 gram://tbn14.nikhef.nl:2119/jobmanager /bin/sh -c
 "/bin/hostname > /tmp/a.txt"
SAGA(NoSuccess): globus_gram_job: Unable run the job because:
[Globus GRAM] the job manager failed to open stderr (74)
```

According to a bug report if a saga application which calls run() on a GRAM job is behind a firewall and can't open stdin/stderr, the application hangs for some time. After that it quits (correctly) with the above mentioned error message.
If we make sure that there is no firewall, the following message is obtained:

```
[tbalint@asen tests]$ saga-job run gram://tbn14.nikhef.nl:2119/
 /bin/sh -c "/bin/hostname > /tmp/hname"
SAGA(NotImplemented): globus_gram_job: get_stdin() is not
 implemented since input streams are supported by GRAM.
```

But in this case the output can be retrieved with *saga-file copy* or *saga-file cat*.
When trying to run a job from a program, first the job description has to be defined, then a job service has to be associated with the current session and a job has to be created. Then the job can be run and the status can be checked.

```
saga::job::description jd;
jd.set_attribute(
 saga::job::attributes::description_executable,
 "/bin/hostname");
jd.set_attribute(
 saga::job::attributes::description_working_directory, "/tmp");
jd.set_attribute( saga::job::attributes::description_output,
 "/tmp/saga_out.txt");
jd.set_attribute( saga::job::attributes::description_error,
 "/tmp/saga_err.txt");
saga::job::service js (my_session, saga::url
 ("gram://tbn14.nikhef.nl:2119/jobmanager"));
saga::job::job job = js.create_job (jd);
job.run();
job.wait();
saga::job::state state = job.get_state();
```

If the standard output and standard error isn't redirected, the output and the error files can't be retrieved. Even is the working directory is set, the output not being redirected, it can't be retrieved.

The arguments of a job can be specified in the following way:

```
std::vector <std::string> args;
args.push_back("+m";
saga::job::description jd;
jd.set_attribute(
 saga::job::attributes::description_executable, "/bin/date");
jd.set_vector_attribute(
 saga::job::attributes::description_arguments, args);
jd.set_attribute(
 saga::job::attributes::description_working_directory, "/tmp");
jd.set_attribute( saga::job::attributes::description_output,
 "/tmp/saga_out.txt");
```

```
jd.set_attribute( saga::job::attributes::description_error ,
  "/tmp/saga_err.txt");
```

This job will successfully runs, the only observation is that if we don't change the file where we redirect the standard output of the job than the output is appended to the existing file (so the file is not recreated).

In case the standard input has to be redirected, the following need to be set in the job description

```
std::vector <std::string> args;
args.push_back("/tmp/saga_in.txt");
jd.set_vector_attribute(
  saga::job::attributes::description_arguments , args);
jd.set_attribute( saga::job::attributes::description_input ,
  "/tmp/saga_in.txt");
```

We also need to specify the file transfer:

```
std::vector <std::string> ft;
ft.push_back ("/tmp/out.txt < /tmp/saga_out.txt");
//retrieve output
ft.push_back( "file://localhost/tmp/saga_in.txt >
  gsiftp://tbn14.nikhef.nl/tmp/saga_in.txt");
//tranfer input
```

When trying to transfer the input we get a Segmentation fault, when only trying to get the output automatically I get

```
*** glibc detected *** ./jobGlobus: free(): invalid pointer:
  0xb4c21e7c ***
```

Remark: It is annoying that there is no command provided to see the job description which is submitted.

According to a bug report:
Globus adaptor _should_ support stage-in/stage-out now:

```
std::vector <std::string> transfers;
transfers.push_back ("file://tmp/FileA > myFileA");
transfers.push_back ("file://tmp/FileB < myFileB");
jd.set_vector_attribute("FileTransfer", transfers);
```

Translates to the following RSL string:

```
(file_stage_in=(file://tmp/FileA
  myFileA))(file_stage_out=(file://tmp/FileB  myFileB))
```

No further URL checking is implemented. The "FileTransfer" attributes have to be specified in a way that GRAM understands them. Furthermore,Globus doesn't return an error if the files don't exist.

Bibliography

[1] GT 4.0 Migrating Guide for WS GRAM. [Online] `http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Migrating_Guide.html`.

[2] Netbeans Profiler. [Online] `http://profiler.netbeans.org/`.

[3] Open Grid Forum. [Online] `http://www.gridforum.org/`.

[4] The DAS 3 project. [Online] `http://www.cs.vu.nl/das3/index.shtml`.

[5] C. Aiftimiei, P. Andreetto, S. Bertocco, S. Dalla Fina, A. Dorigo, E. Frizziero, A. Gianelle, M. Marzolla, et al. Design and implementation of the gLite CREAM job management service. *Future Generation Computer Systems*, 2009.

[6] C. Aiftimiei, P. Andreetto, S. Bertocco, S. Fina, S. Ronco, A. Dorigo, A. Gianelle, M. Marzolla, M. Mazzucato, M. Sgaravatto, et al. Job submission and management through web services: the experience with the CREAM service. In *Journal of Physics: Conference Series*, volume 119, page 062004. IOP Publishing, 2008.

[7] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *Global Grid Forum GFD-RP*, 20, 2003.

[8] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. Van Nieuwpoort, A. Reinefeld, F. Schintke, et al. The grid application toolkit: toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[9] G. Allen, T. Goodale, T. Radke, M. Russell, E. Seidel, K. Davis, K. Dolkas, N. Doulamis, T. Kielmann, A. Merzky, et al. Enabling applications on the grid: a gridlab overview. *International Journal of High Performance Computing Applications*, 17(4):449, 2003.

[10] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424. IEEE, 2004.

[11] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. Schopf, M. Viljoen, and A. Wilson. GLUE schema specification.

[12] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job submission description language (jsdl) specification, version 1.0. In *Open Grid Forum, GFD*, volume 56, 2005.

[13] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, et al. Web services security (WS-Security). *IBM developerWorks, [Online]* `http://www-106.ibm.com/developerworks/library/ws-secure`, 2002.

[14] J. Brittain and I. Darwin. *Tomcat: the definitive guide*. O'Reilly Media, Inc., 2007.

[15] S. Burke, S. Campana, A. Peris, F. Donno, P. Lorenzo, R. Santinelli, and A. Sciaba. CERN, GLITE 3 User Guide, 2007.

[16] J. Ceriel and T. Kielmann. A simple API for Grid applications (SAGA) - Java Language Bindings, 2008.

[17] K. Channabasavaiah, K. Holley, and E. Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16, 2003.

[18] S. Chiba. Load-time structural reflection in Java. *ECOOP 2000âĂŤObject-Oriented Programming*, pages 313–336, 2000.

[19] R. Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana. Web services description language (WSDL) version 1.2 part 1: Core language. *World Wide Web Consortium, Working Draft WD-wsdl12-20030611*, 2003.

[20] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-resource framework. 2004. In *Global Grid Forum*.

[21] K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. Modeling and managing state in distributed systems: The role of OGSI and WSRF. *Proceedings of the IEEE*, 93(3):604–612, 2005.

[22] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 1998.

[23] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. Foster. Grid Information Services for Distributed Resource Sharing. *High-Performance Distributed Computing, International Symposium on*, 0, 2001.

[24] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 131–140. Springer, 2004.

[25] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[26] A. Elwell. LCG Computing Element (LCG CE). [Online] `https://twiki.cern.ch/twiki/bin/view/EGEE/LcgCE`, 2010.

[27] T. Erl. *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

[28] D. Fallside, P. Walmsley, et al. XML schema part 0: Primer. *W3C recommendation*, 2:0–20010502, 2001.

[29] I. Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[30] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.

[31] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, volume 22, pages 1–5. Edinburgh, 2002.

[32] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92. ACM, 1998.

[33] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.

[35] A. Gianoli, K. Lorentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In *Grid computing: first European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2003: revised papers*, page 33. Springer-Verlag New York Inc, 2003.

[36] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A simple API for Grid applications (SAGA). In *Global Grid Forum Document GFD*, volume 90, 2007.

[37] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology*, 12(1):7–20, 2006.

[38] M. Gudgin, M. Hadley, N. Mendelsohn, Y. Lafon, J.-J. Moreau, A. Karmarkar, and H. F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C recommendation, W3C, Apr. 2007. Retrieved from http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[39] H. Haas and A. Brown. Web services glossary. *W3C Working Group Note*, 11:2009–08, 2004.

[40] D. Hollingsworth et al. Workflow management coalition: The workflow reference model. *Workflow Management Coalition*, 1993.

[41] V. Huber. UNICORE: A Grid computing environment for distributed and parallel computing. *Parallel Computing Technologies*, pages 258–265, 2001.

[42] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl 2):W729, 2006.

[43] H. Kaiser, A. Merzky, S. Hirmer, and G. Allen. The SAGA C++ Reference Implementation. In *2nd Int. Workshop on Library-Centric Software Design (LCSDâĂŹ06)*. Citeseer, 2006.

[44] V. Korkhov, A. Belloum, and L. Hertzberger. VL-E: Approaches to Design a Grid-Based Virtual Laboratory. *Distributed and Parallel Systems*, pages 21–28, 2005.

[45] V. Korkhov, A. Wibisono, D. Vasyunin, and A. B. et al. Vlam-g: Interactive data driven workflow engine for grid-enabled resources. *Scientific Programming*, 15(3):173–188, 2007.

[46] D. Koufil and J. Basney. A credential renewal service for long-running jobs. In *The 6th IEEE/ACM International Workshop on Grid Computing, 2005*, page 6, 2005.

[47] M. Lorch, J. Basney, and D. Kafura. A hardware-secured credential repository for grid PKIs. In *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004*, pages 640–647, 2004.

[48] C. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference model for service oriented architecture. *OASIS Committee Draft*, 1, 2006.

[49] J. Mayer, I. Melzer, and F. Schweiggert. Lightweight plug-in-based application development. *Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 87–102, 2009.

[50] P. Mishra, E. Maler, C. Cahill, A. Hughes, A. Origin, M. Beach, B. Metz, B. Hamilton, R. Randall, A. Booz, et al. Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2. 0. *Language (SAML)*, 2:0, 2005.

[51] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 50–59. IEEE, 2002.

[52] H. Rajic, I. Inc, W. Chan, I. Ferstl, A. Haas, and J. Tollefsrud. Distributed resource management application API specification. In *Global Grid Forum*, 2002.

[53] S. Reynaud. Uniform access to heterogeneous grid infrastructures with JSAGA. *Production Grids in Asia*, pages 185–196, 2010.

[54] M. Riedel, A. Streit, D. Mallmann, F. Wolf, and T. Lippert. Experiences and Requirements for Interoperability Between HTC and HPC-driven e-Science Infrastructure. *Future Application and Middleware Technology on e-Science*, pages 113–123, 2010.

[55] J. Shirazi. *Java performance tuning*. O'Reilly Media, Inc., 2003.

[56] J. Snell, D. Tidwell, and P. Kulchenko. *Programming Web services with SOAP*. O'Reilly Media, 2002.

[57] A. S. Tanenbaum and M. v. Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., 2007.

[58] I. Taylor. Triana generations. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, page 143. IEEE, 2006.

[59] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[60] P. Troger, H. Rajic, A. Haas, and P. Domagalski. Standardization of an api for distributed resource management systems. In *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007*, pages 619–626, 2007.

[61] S. Tuecke, W. Welch, D. Engert, L. Pearlman, and M. Thompsom. Internet X. 509 Public Key Infrastructure Certfificate (PKI) Proxy Certificate Profile. Technical report, June 2004. IETF RFC 3820.

[62] R. van Nieuwpoort, T. Kielmann, and H. Bal. User-friendly and reliable grid computing based on imperfect middleware. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing-Volume 00*, pages 1–11. ACM, 2007.

[63] J. Viega, M. Messier, and P. Chandra. *Network security with OpenSSL*. O'Reilly Media, Inc., 2002.

[64] G. Von Laszewski, I. Foster, and J. Gawor. CoG kits: a bridge between commodity distributed computing and high-performance grids. In *Proceedings of the ACM 2000 conference on Java Grande*, pages 97–106. ACM, 2000.

[65] Von Welch and Tom Barton and Kate Keahey and Frank Siebenlist. Attributes, anonymity, and access: Shibboleth and globus. integration to facilitate grid collaboration. In *In 4th Annual PKI R&D Workshop*, 2005.

[66] D. Vudragovic. Simple tomcat-ssl integration and dn-based authentication. [Online] `http://wiki.egee-see.org/index.php/Simple_Tomcat-SSL_integration_and_DN-based_authentication`.

[67] V. Welch, R. Ananthakrishnan, S. Meder, L. Pearlman, and F. Siebenlist. Use of saml in the community authorization service. Global Grid Forum Open Grid Services Architecture Security Working Group [Online] `http://www.globus.org/toolkit/security/ogsa/authz/OGSA-SAML-authorization-profile-june4.pdf`, 2003.

[68] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3):171–200, 2005.

List of Acronyms

BDII Berkeley Database Information Index, 17, 20, 53

CA Certificate Authority, 15, 30, 38, 39, 54, 56, 78, 82, 83

CREAM CE Computing Resource Execution And Management [5, 6], 7, 20, 23, 25, 57, 61–67, 89, 107, 117

DRMAA Distributed Resource Management Application API [52, 60], 21, 22

GAT Grid Application Toolkit [8, 9], 21, 22, 91

GLUE Grid Laboratory Uniform Environment [11], 17, 53

gMInION Grid MIddleware Independent jOb maNager, 27, 36, 43, 44, 48, 57–65, 67, 69, 73, 76, 77

GRAM Grid Resource Allocation and Management, 19, 20, 73, 129, 133

GSI Grid Security Infrastructure, 15, 16, 18, 19, 41, 116

GT Globus Toolkit [30, 29], 14, 40, 73, 74, 78, 85, 107, 116, 129

GUID Grid Universal Identifier, 18

JDL Job Description Language [12], 19, 23, 25, 42, 87, 115, 117, 118, 128

LCAS Local Center Authorization Service, 16

LCG-CE LCG Computing Element [26], 7, 20, 23, 25, 57, 60–62, 65–67, 88

LCMAPS Local Credential Mapping Service, 16

LDAP Lightweight Directory Access Protocol, 16, 17, 20

LFN Logical File Name, 18

MDS Monitoring and Discovery Service, 16, 19, 20

RSL Resource Specification Language [22, 1], 19, 85, 86, 111

RTSM Runtime System Manager, 73, 74

SAGA Simple API for Grid Applications [37, 36], 21–23, 105, 106, 129

SOAP Simple Object Access Protocol [38], 14, 45, 46, 48, 49, 58, 70, 76

SRM Storage Resource Manager, 18, 20, 23, 25, 107, 110, 113, 116, 118, 122

SSL Secure Socket Layer, 38, 39, 82

SURL Site URL, 18

VO Virtual Organization, 16–18, 29, 53, 66, 78

VOMS Virtual Organization Membership Service [35], 15, 16, 20, 42, 78, 107, 112, 116, 126

WMS Workload Management System, 19, 20, 22, 23, 25, 42, 54, 57, 59–67, 69, 72, 76, 87, 107, 110, 112, 114, 117

WSDL Web Service Description Language [19], 14, 45, 48, 49

WSRF Web Services Resource Framework [20, 21], 14, 73