Extending the Java Grid Application Toolkit to Globus Web Services

Balazs Bokodi

July 6, 2007

M.Sc thesis
by
Balazs Bokodi

Supervisor: Thilo Kielmann
Co-supervisors: Kees van Reeuwijk, Rob V. van Nieuwpoort

Contents

Chapter 1

Introduction

In the past 10 years the grid technology has been heavily developed, so these days it is commonly used among different organizations. These organizations execute applications which have huge data or CPU consumption. A couple of massive grid services are accessible for the members of the scientific society today. Our university has a cluster site, which is part of the DAS-3 grid. The DAS-3 supercomputer connects five cluster sites together and contains more than 250 nodes.

In spite of the spreading of grid computing, developing application for grids is troublesome, because grid computing raises a lot of problems. To solve these problems new solutions come out every day and that progress has a very unfavorable influence for application developers: they have know the aspects of the latest grid middleware.

The other difficulty for grid developers is the wide choice of grid middlewares. Similar services are offered by many grid middleware manufacturers, and the programming interfaces are not compatible.

To make the life of the programmers easier higher-level abstractions have been established. Java Grid Application Toolkit (JavaGAT) is such a high level grid application toolkit, and in this thesis we show how to adapt that toolkit for Globus 4 grid middleware.

1.1 Problem statement

This thesis deals with a quite practical side of the grid computing. The interface offered by Java Grid Application Toolkit (JavaGAT) is grid-application oriented and it is designed especially for grid programmers, who want to create rapidly the proper application without bothering with the underlying low-level mechanisms. The structure and purpose of a grid API different from JavaGAT. The the primary aim of a grid API is to wrap and to use the middleware services. For this reason the adaptation of Globus 4 raises many problems. This document wants to answer the following questions:

- What interfaces or APIs do exist for Globus 4?
- Which toolkit or API can be used to adapt JavaGAT to Globus 4?
- How can we implement an adaptor using these toolkits or APIs?

- How efficient is the implementation of the Globus 4 adaptors?

Some other related sub-questions:

- What is new in the implementation of the Globus 4 toolkit?
- How is the JavaGAT built up?

Our work has started with a small task: implementation of file adaptors for JavaGAT. We decided to create Samba adaptors for JavaGAT. This thesis answers the following questions related to the Samba adaptors:

- How can we implement file adaptors for JavaGAT?
- How can be an adaptor tested?
- How efficient is the implementation of our Samba adaptor?

1.2 Outline of the thesis

- Chapter 1 of the thesis gives a briefly introduction about my project itself and describes the structure of the document.
- Chapter 2 is intended to discuss the related work.
- In chapter 3 we can see the Globus 4 Toolkit. The services, the structure and native APIs of toolkit is explained.
- Chapter 4 shows the JavaCoG kit, that API is used to develop grid applications for Globus.
- The JavaGAT is represented in chapter 5. The purpose and usage of the JavaGAT is described there.
- After that the Samba adaptor is disscussed in chapter 6. In that chapter we share the experience how an adaptor is implemented for the JavaGAT.
- In chapter 7 we explain the details of our solutions. The result of the performance measurements are also represented there.
- In chapter 8 we take a comprehensive look on the thesis, and the results are evaluated and analyzed.

Chapter 2

Related work

This chapter reviews shortly the related work on the JavaGAT adaptor.

2.1 Related work

Globus adaptors for JavaGAT have already been written. This implementation uses the pre-WS Globus API. In the later sections the differences between the new Globus 4 and the old Globus adaptors and the performance of this adaptors are compared.

The Globus adaptor for JavaGAT has been created by Rob V. van Nieuwpoort. This implementation uses the jGlobus interface of JavaCoG kit.

Chapter 3

Globus 4

The Globus Toolkit[1] is an open source software toolkit used for building a grid. The grid lets people share computing power, databases, and other tools securely on-line across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management.

The Globus Toolkit is being developed by the Globus Alliance and many others all over the world, and it is one of the most popular software package for creating grids. A growing number of projects and companies use the Globus Toolkit to unlock the potential of grids for their target.

The other reason for popularity of the Globus Toolkit is that it meets the Open Grid Services Architecture (OGSA). OGSA is developed by The Global Grid Forum, which defines a standardized and open architecture for the grid-base applications. These set of requirements are a sort of de facto standard of the grid-community.

In this section the Globus 4 Toolkit is briefly described with its most remarkable innovations. Some of these new developments are detailed from the point of the Globus 4 adaptors with the related backgrounds.

3.1 Globus 4 toolkit

This section explains the naming, the services and the architecture of the Globus 4 toolkit.

The GT4 stands for version 4 of the Globus Toolkit. In this version significant innovations were introduced. The most important is that the majority of the services is built on the top of the WSRF (Web Service Resource Framework). The Web Service based components of the Globus Toolkit are referred as WS components while the rest are the pre-WS or non-WS components.

The components of the Globus Toolkit is represented on the figure 3.1[2]. On the top of the diagram we can see the WS components, below that the non-WS components are shown. The category of the components can be security, data

[1] http://www.globus.org/toolkit/

[2] origin of the figure: http://www.globus.org/toolkit/about.html

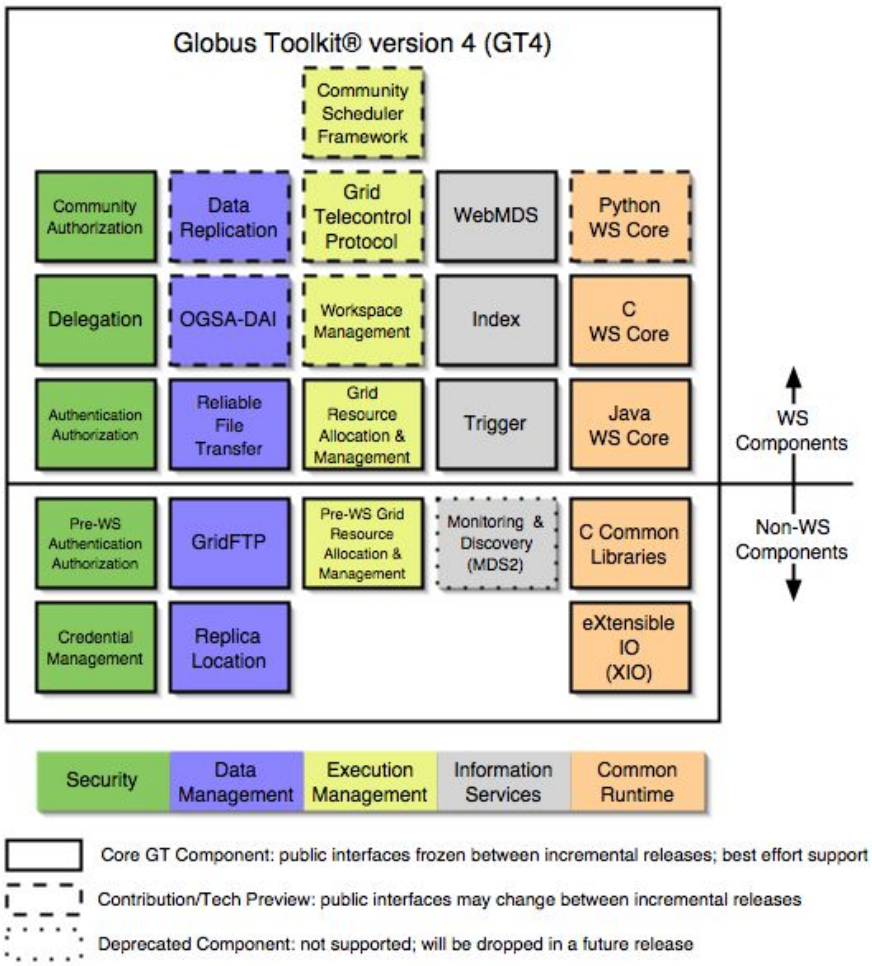

Figure 3.1: GT4 components

management, execution management, information services, common runtime. The components on the picture are coloured by the categories.

The services of Globus toolkit are collected into these categories:

Security The Grid Security Infrastructure (GSI) uses the public key cryptography. The concept of GSI authentication is the certificate. Every user and service on the grid is identified with its certificate. Since we use public key cryptography, the private key of the user has to be protected. Usually the private key is secured by a passphrase. Furthermore the grid applications should act on behalf of the user. Therefore these long running grid application needs that private key to authenticate itself against the services. Instead of asking the user's passphrase again and again, Globus has introduced delegation and proxy certificates.

Data management Data management can be devided into two categories: data movements and data replication. Two components related to the data movement: GridFTP and Reliable File Transfer (RFT). The GridFTP protocol is not a web service protocol, whereas RFT is a WSRF compliant web service. Data replication is not discussed in this document. The GridFTP is based upon the Internet FTP protocol and implements extensions for high-performance, secure, reliable data transfers.

Execution management The Globus Toolkit provides two suites to submit, monitor and cancel jobs on grid resources. Both systems are known under the moniker "GRAM", whereas "WS GRAM" refers only to the web service based one.

Information services The Monitoring and Discovery System (MDS) allows users to discover what resources are considered as part of a Virtual Organization (VO) and to monitor those resources. That system is not discussed in this document.

Common runtime The common runtime components are a set of libraries and tools which needed to provide the GT4 web services and the pre-web services. Platform independent abstraction layers are built with the common runtime components and the functionality lower in the web services stack are leveraged.

3.2 Web Services

As mentioned earlier, the most significant change in the Globus 4 toolkit is the Web Service Resource Framework (WSRF). The WSRF is a joint effort by the Grid and the Web Service communities, so it fits into the Web Service architecture. In this section first the motivation of the change to WS is discussed then the application of it is explained.

The Web Service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable Machine to Machine interaction over the network. In common usage the clients and servers communicate using XML messages that follow the Simple Object Access Protocol (SOAP).

The WSRF extends the web services. The main advantages of web services over other technologies:

- Web Services are platform-independent and language-independent, since they use standard XML language. This means the client programs can be programmed and run on one platform and language while the Web Services can be developed and placed in different environment.
- The Web Services usually use HyperText Transfer Protocol (HTTP) for transmitting messages. This is a major benefit for building Internet-scale application, because the Internet proxies and firewalls will not interfere with HTTP traffic.

The web services allow us to create client/server applications. A typical web service invocation consists of the following steps (after that list the applied protocols are explained):

1. The location of the service is known or unknown. If it is unknown we are going to discovery a service that meets with our requirements through a discovery service (which is itself a web service).
2. The discovery service replies and tells us which server provides the requested service.
3. The client asks the server to describe itself.
4. The server replies with a description of itself.
5. The location of the Web Service and how to invoke it is known, so the client sends a SOAP request.
6. The Web Service replies a SOAP response which includes the answer or an error message.

These are the essential parts of the Web Service architecture:

Service Processes The discovery, the aggregation, the choreography or any other processes belong to that part of the architecture which processes involve generally more than one Web Service. For example the discovery fits here, because it allows us to locate one particular service from among a collection of Web Services.

Service Description The Web Services are self-describing. That means once the web service is located it can be asked what operations it supports and how to invoke them. This is handled by the Web Services Description Language (WSDL).

Service Invocation Invoking a Web Service works through by passing messages between the client and the server. The Simple Object Access Protocol (SOAP) specifies how the request to the server should be formatted, and how the server should format its responses. SOAP is based on XML messages and the messages normally are sent over the network using HTTP/HTTPS.

Transport The HTTP (HyperText Transfer Protocol) is used to transmit messages between the client and the server, the same protocol used to access conventional web pages on the Internet.

Web service addressing: the web services are addressed just like web pages. The web services are accessible via URI (Uniform Resource Identifier). An address might look like this:

```
https://fs0.das3.cs.vu.nl:8443/wsrf/services/ManagedJobFactoryService
```

As we can see web services provide us a low-level, versatile, standardized form of client-server communication. However, this protocol is still stateless, which means the invocation of the server from one client is independent, and this issue makes difficult the design of complex distributed systems.

The web services are stateless, that means the invocations of the web service are indepentenly called. The web service does not store any information of the calls. For this reason a web service cannot act like a counter or register.

3.3 Web Service Resource Framework

To build up grid applications we need stateful services. The WSRF uses a simple solution: it keeps the web services and the state information completely separate. For example an accumulator service can use resources to store its state. The resources are integers in that case as it is shown in Figure 3.2[3].

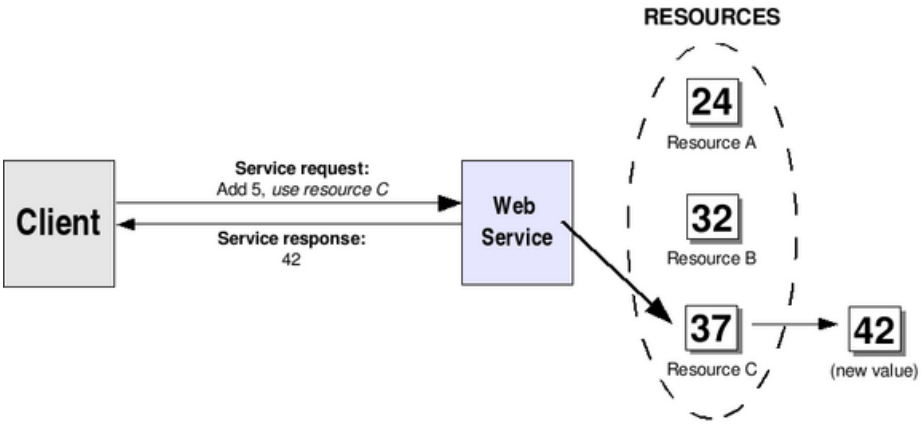

Figure 3.2: Stateful accumulator service

The web service keeps the state in a separate entity called *resource*, which stores all the state information. Each resource has a identity and lifecycle. Whenever we want a stateful interaction with the server we simply instruct the web service to use a specified resource. To achieve this the web service has to be told which resource we want to use. Therefore the web service addressing is expanded with a WS-Resource part.

To manage resources the Web Services Resource Framework comes with five specifications:

WS-Resource defines WS-Resource as a composition of a resource and a Web Service. The resource is accessible through the Web Service.

[3]origin of the figure: http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html

WS-ResourceProperties describe the interfaces typed values which values are associated with a WS-Resource. Theses values might be read and manipulated in a standard way.

WS-ResourceLifetime describes how to manage the lifetime of a WS-Resource, since the resources are not static entities.

WS-ServiceGroup describes an interface for operating on collections of WS-Resources or on groups of Web Services.

WS-BaseFaults specification defines the standard mechanisms to reporting faults when something goes wrong during the Web Service invocation.

There are two other specifications which are not part of the WSRF specification, and are also relevant. One is the **WS-Notification** which says how web services have to be configured as a *notification producer*, and certain clients to be *notification customer*. That means if a change occurs in one of the WS-Resources, the change is notified to all the subscribers. This specification allows web services to push information.

The other necessary specification is the WS-Addressing, that makes possible a more versatile URI than the plain ones. It consists of two parts: a structure for communicating with a Web Service endpoint (the specified resource is encapsulated in the endpoint reference), and a set of Message Addressing Properties. We can use WS-Addressing to address (Web Service + a WS-Resource) pair.

3.4 Globus API

The Globus Toolkit 4 provides a Java and a C API. The only relevant interface for our interest is the Java API. Unfortunately the introduction of the Web Services in the GT4 affects the grid application developers, since the changes in API could not be avoided at this level, although lot of pre-WS implementations are kept from the previous versions. The GT4 API provides an interface for the WS-based grid component, which means new libraries and a new programming models for the programmers.

The usage of the Globus API is detailed in section 7.3 . Here we can see the scenario of an invocation of a generalized GT4 service:

1. First an `EndpointReferenceType` object is created representing the endpoint reference of a service. This endpoint reference is used to address a particular WS-Resource.

2. Then we obtain a reference to the `portType` of the service. That object will allow us to contact the given service.

3. Once we have that reference we can work with the Web Service. The operations can be invoked just like a local method calls on the object.

4. The code must be placed inside the catch/try block, since all of these operations are remote calls and `RemoteException` can be thrown.

These steps are used to carry out file transfer or job submission in GT4 using WS-services, while the non-WS services have a particular interface to execute operations.

Chapter 4

JavaCoG kit

The Commodity Grid (CoG) Kits[1] allow Grid users, application developers, administrators to use, program and administer Grids from a high level framework. The JavaCoG is an open source contribution software to Globus Toolkit from the CoG group. Its leader developer is Gregor von Laszewski, who works at the Argonne National Laboratory. The jGlobus module of the CoG kit is distributed with the Globus Toolkit as a part of the default installation. The framework has different levels of abstractions to use Grid services or to develop grid applications. The API levels protect the programmer from the changes of the underlying Grid middleware.

If we want to wrap grid services the important layers are the jGlobus library (called APIs on the figure 4.1) and the JavaCoG Kit abstractions. These layers, as the figure[2] shows, can exploit the functions of the grid. The structure of the CoG kit is represented on the figure 4.2.

The higher levels support workflow designing and monitoring, these levels are not relevant in the context of this thesis.

4.1 The jGlobus library

JavaCoG Kit JGlobus module gives a core Java API to access GridFTP servers, the non-WS GRAM services and a complete implementation of GSI. The JGlobus module is one jar file, called `cog-jglobus.jar`, which is distributed with GT3 and GT4. The other layers heavily rely on the classes and methods implemented in jGlobus.

JavaGAT already has a set of adaptors for Globus, which are implemented with the JGlobus API. Because GT4 has both WS and non-WS services to data management and GRAM, a complete set has been written to wrap the I/O function, and the resource broker works for GT4. The I/O adaptors are file, input and output stream adaptors for GridFTP. The resource broker adaptor makes available the non-WS GRAM services for the grid application developers by the JavaGAT interface.

[1] http://wiki.cogkit.org/index.php/Main_Page
[2] origin: http://wiki.cogkit.org/index.php/Java_CoG_Kit

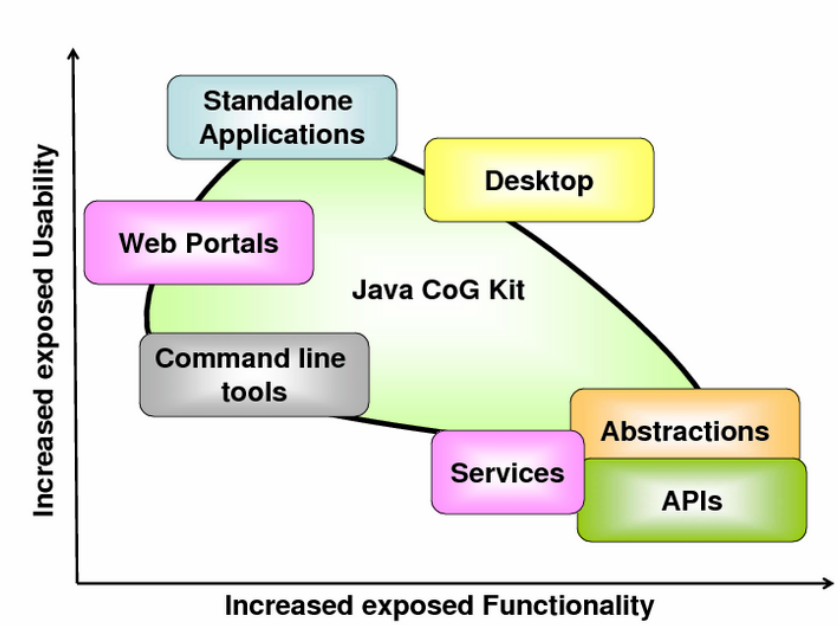

Figure 4.1: Integrated approach

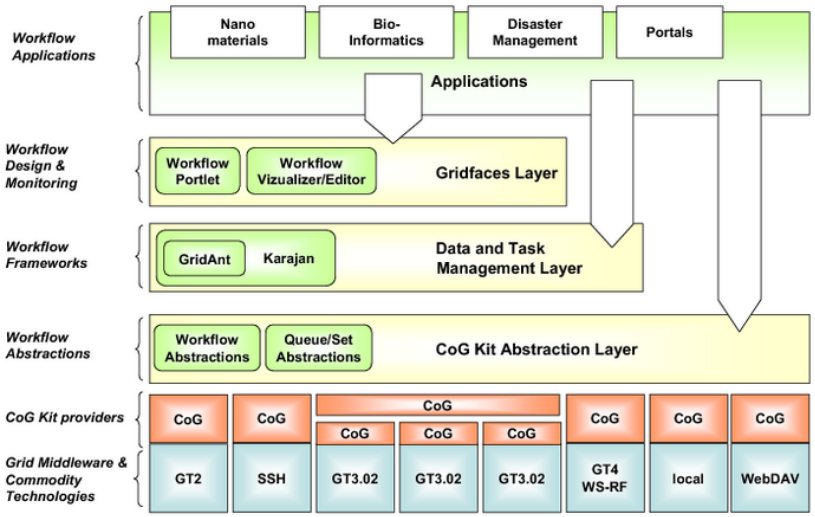

Figure 4.2: Layered Architecture

4.2 JavaCoG abstractions

JavaCoG Kit abstractions offer a programming model that supports pattern for job execution, file transfer and file operations. Although not relevant for us, it also supports advanced job execution for directed acyclic graphs. The job execution pattern has providers for GT v2.4.3, GT v3.0.2, GT v3.2.0, GT v3.2.1, GT v4.0.0, Condor, and SSH implementations. If we look at section 5.1, the purpose and architecture of the JavaCoG abstractions is close to JavaGAT.

JavaCoG API has a uniform API which is independent from the grid middleware, and its primary aim is to support the different versions of the Globus. The strengths of the toolkit are the abstraction and the provider model.

The provider model makes for developers runtime selection of the grid services possible. The dynamically selected provider uses the appropriate grid service that carries out the job submission or the file transfer. This capability uses customized dynamic class loading and late binding against the existing grid middleware. This provider model accords to the use of adaptors in the JavaGAT. (The JavaGAT philosophy is shown in the section 5.1.) The providers and their location in the layer-hierarchy are represented in Figure 4.2[3].

The Java Cog Kit abstractions offers a programming model that supports elementary Grid patterns such as job execution, file transfer and file operations. This paragraph represents this programming model. This model is essential to know if we want to implement an adaptor.

In the next list we can see the steps of job submission.

- First a `Task` object should be created and set with appropriate provider. The default option for the provider can be GT2, GT3.0.2, GT3.2.0, GT3.2.1, GT4.0.0, Condor or SSH. For the full list of possible values of providers see appendix A.1.
- Second the `JobSpecification` object is created with the appropriate properies. Then the `JobSpecification` is passed to the `Task`.
- In the following step we create a service to the task. Provider, service contact and security context are assigned to the service, after that the `Service` object is passed to the `Task`.
- This step is optional, but usually desired. We can implement a `StatusListener` class to monitor the status of our task.
- In the last step the task can be submitted for execution using a `TaskHandler` object.

This task model is also applied for file transfers. The major difference is that instead of the `JobSpecification` the `FileTransferSpecification` class is used, which allows us to set up source and destination URLs.

Two ways are given to execute file operation. One uses the task model, as the file transfer and task execution. The additional item is that before execution a file operation, a session identity has to be acquired. The session ID can be retrieved by execution of the `FileOperationSpecification.START` operation. The second option to carry out file operation is a more convenient solution for the application developers. That model does not adhere the task model. We only

[3]origin of the figure: http://wiki.cogkit.org/index.php/Java_CoG_Kit

need to create a `FileResource` then the invocation of file operations is done by calling the appropriate methods of the object. Of course the security context and the service contact must be passed and the object has to be initialized by `start` method.

Chapter 5

JavaGAT

Java Grid Application Toolkit (JavaGAT) is represented in this section. First we take a general view of JavaGAT. Then the structure of it is detailed with some additional information for developers.

JavaGAT[1] is part of The GridLab Project[2]. The GridLab project is one of the biggest European research undertakings in the development of application tools and middleware for Grid environments. The JavaGAT is an open source abstraction layer above the grid middlewares, and it is written in Java, so it is highly portable. The applications can be compiled and linked on any machine without available grid service, then it can run in grid environment. Theoretically, the applications can be tested anywhere. The toolkit provides generalized grid functions that grid application programmers need. These functions are connected to security, Grid I/O, resource management, application information management and monitoring.

5.1 The Java Grid Application Toolkit

The toolkit carries out the operations independently of the underlying grid infrastructure. To access the different grid infrastructures JavaGAT uses adaptors. Separated adaptors are used to connect different grid services. Multiple adaptors can provide the same functions. Technically the adaptors are Java JAR files and are dynamically loaded into the Application. JavaGAT also supports late binding. That means the GAT engine provides the runtime delegation of GAT-API calls to adaptors. The GAT engine dynamically loads the adaptors and tries to choose the most applicable adaptor. The engine determines which adaptor can provide the operations. It chooses one adaptor, then if the adaptor fails the engine might try another one.

JavaGAT can be separated into two parts. One is the JavaGAT engine which is responsible for providing the interfaces for the programmers and handles the adaptors. The other part of JavaGAT are the adaptors. Let the `$GAT_LOCATION` be the location of JavaGAT in our file system. So the sources and binaries of the engine are placed under the `$GAT_LOCATION/engine` directory and the sources and binaries of the adaptors can be found at `$GAT_LOCATION/adaptors` location.

[1] https://gforge.cs.vu.nl/projects/javagat/
[2] http://www.cs.vu.nl/ibis/javagat.html

Two other subdirectories can be seen at the root of JavaGAT, the first is the `$GAT_LOCATION/bin` which contains shell scripts to invoke the Java application based on the JavaGAT. The second is `$GAT_LOCATION/tests` where the applications can be put.

JavaGAT engine, adaptors and tests are compiled by `ant` Java build tool. The building information can be found in the `$GAT_LOCATION/build.xml` file. That file relies on other `build.xml` files, which are located in the adaptors, engine and tests subdirectories.

The JavaGAT API (both engine and adaptors) can be categorized into these sections, and the structure of the Java packages follows this system:

- Security: deals with username-password pairs or with credentials.
- Grid I/O: covers File operation, remote file access, file replication and interprocess communication.
- Resource Management: wraps resource brokering, forking grid applications and job management.
- Application Information Management: has a global repository for application specific information, queries.
- Monitoring: includes the grid monitoring and application monitoring and steering.

So the package hierarchy is trailed by that structure as well. Based on the function of the class it is put under the `org.gridlab.gat.security`, `org.gridlab.gat.io`, `org.gridlab.gat.resources`, `org.gridlab.gat.advert` and `org.gridlab.gat.monitoring` path. The directory structure also goes after the package hierarchy.

The adaptors are developed simply by extending and implementing the methods of a Capability Provider Interface (cpi) class. We can find the cpi classes under the location of the engine at the appropriate location depending on the purpose of the class. For example the FileCpi.Java file path is `$GAT_LOCATION/engine/org/gridlab/gat/io/cpi/FileCpi.java`.

Chapter 6

Creation of the samba adaptors

In this chapter we can see how an adaptor is implemented for JavaGAT. First the problem is clarified, then the details of the implementation comes, finally the performance of the adaptor is analyzed.

The samba adaptors for Globus are not related to the Globus 4 adaptor. This task was a practise to get known with JavaGAT. For that reason it is mentioned in this document, as well.

6.1 JavaGAT I/O adaptor for Samba

The task is to design and to create Samba adaptor for JavaGAT.

The JavaGAT I/O API offers the following classes for the programmers:

- `File`, `FileInputStream`, `FileOutputStream`, `RandomAccessFile`. These classes extend the generic `java.io` classes.
- `LogicalFile`: replicated file support.
- Inter-process communication: `Endpoint`, `Pipe`, `PipeListener`.

The Samba adaptors support the functions of the `File`, `FileInputStream` , `FileOutputStream` and `RandomFileAccess` classes. To enable these classes for samba, some abstract classes of the JavaGAT engine have to be extended. For the verification and testing of the adaptor a test suite is created.

6.2 Samba

Samba is an open source/free software suite supports seamless file and print services to SMB/CIFS clients. SMB stands for Server Message Block, applied to shared access for files, printers, serial ports and miscellaneous communications between nodes. SMB is mainly used by Microsoft Windows equipped computers. The CIFS is the Common Internet File System, it was renamed from SMB in 1996, and Microsoft added more features like handling symbolic links, hard links, larger file supporting, direct connection without all NetBIOS trimmings.

6.3 Programming environment

The JavaGAT source can be downloaded from a website of our university[1]. To manage the code the svn version control system is used. The detailed description of the environment settings can be seen in the appendix B.1.

6.4 Implementation details

We use the Java CIFS Client Library (JCIFS) to implement the file operations that are needed to execute on the Samba file system. Those interfaces and the JavaGAT I/O interfaces accord to the Java I/O classes, therefore the implementation of the adaptor can be plainly coded. The I/O cpi classes of the GAT have to be extended and the build.xml should be modified. Finally some test applications have been created.

JCIFS is an Open Source client library that implements the CIFS/SMB networking protocol in 100% Java. Since the JCIFS[2] library is used, it is put into the `$GAT_LOCATION/adaptors/external` and `$GAT_LOCATION/tests/external` directories. It is only one jar file named jcifs-1.2.13.jar.

6.5 Tests

Small programs are created to test and verify the Samba adaptor. These files are placed in the `$GAT_LOCATION/test/myprobe` directory.

Files and their functions:

- `SmbFile.java`: it can test JavaGAT `File` class for these operations: `list`, `listFiles`, `exists`, `isDirectory`, `length`, `mkdir`, `mkdirs`, `delete`, `canRead`, `canWrite`, `createNewFile`, `isFile`, `isHidden`, `lastModified`.
- `SmbRAFile.java`: it is for testing `RandomAccessFile` object. It do some read and write from the given file.
- `SmbInputStream.java`, `SmbOutputStream.java`. The `SmbInputStream.java` reads the content of a given file and writes it to the standard output. The `SmbOutputStream.java` writes a "Hello world!" string into the given file.

The command line invocation and usage of these applications can be seem in the appendix B.2.

6.6 Results

Our benchmarks consist of two part. One benchmark measures the speed of the input/output streams, while the other one evaluates the file copy.

The benchmars of streams are file copies, as well. Four implementations of file copy are used to compare the performance of the I/O stream adaptors. Two applications measure the speed of the JavaGAT streams. These two applications are created with `FileInputStream` and `FileOutputStream` of `org.gridlab.gat.io`

[1] https://gforge.cs.vu.nl/projects/javagat/
[2] http://jcifs.samba.org/

and `java.io packages`. The use of the streams are these (file copy from input stream → output stream):

- `CpGATToLocal`: `org.gridlab.gat.io.FileInputStream` → `java.io.FileOutputStream`
- `CpLocalToGAT`: `java.io.FileInputStream` → `org.gridlab.gat.io.FileOutputStream`

The other two applications measure the speed of the native java implementation of the samba streams:

- `CpSambaToLocal`: `jcifs.smb.SmbFileInputStream` → `java.io.FileOutputStram`
- `CpLocalToSamba`: `java.io.FileInputStream` → `jcifs.smb.SmbFileOutputStream`

The time of the file copy has been measured. We check the start time before the reading of the first block, and the clock is stopped after the the last block is written. The creation of the stream objects and initialization are not included in time. The call of the programs from command line can be seen in appendix B.4. The tests have been done on a single node installed JavaGAT and Samba (see appendix B.3).

The results of the benchmarks are represented in Figures 6.1, 6.4, 6.2, 6.3. The first cells of the columns refer the stream used by the implementation that we measured. (For example JavaGAT, sftp is the JavaGAT stream using the sftp adaptor.) We have done three measurements for each case. The results of these measurements are shown below in the first row in milliseconds. Then we can see the average for each measurement. Below that the speed is converted into MB/sec, as well.

The results of the copy with input stream of JavaGAT and jcifs(`org.gridlab.gat.io` and `jcifs.smb`) to output stream of `java.io`, with file size 50 MB are shown in the table 6.1. JavaGAT implemented stream is slower than the native `jcifs.smb` implementation (JavaGAT: 5 MB/sec and the `jcifs.smb` 28 MB/sec).

	JavaGAT, local	JavaGAT, sftp	JavaGAT, smb	jcifs.smb
1. run	838	5717	8623	1769
2. run	807	14011	8369	1747
3. run	772	9854	8243	1687
average	805	9860	8411	1734
MB/sec	62	5	5	28

Figure 6.1: The results of the input stream measurements with 50MB file size

On the figure 6.2 we can see the results of the copy from input stream of `java.io` to output stream of `jcifs.smb` and `org.gridlab.gat.io` with 50 MB file size. The speed difference of JavaGAT and `jcifs.smb` implementation is less than in the previous benchmark (JavaGAT: 6 MB/sec and `jcifs.smb` 19 MB/sec).

The result of the copy from the input stream of `jcifs.smb` and `org.gridlab.gat.io` to the output stream of `java.io` with file size of 100 MB is represented on the figure 6.3. The change of the speed beetween JavaGAT and `jcifs.smb`

	JavaGAT, local	JavaGAT, sftp	JavaGAT, smb	jcifs.smb
1)	851	37596	8135	2506
2)	805	39411	8125	2642
3)	814	38365	8142	2560
average	823	38457	8134	2569
MB/sec	60	1	6	19

Figure 6.2: The results of the output stream measurements with 50MB file size

	JavaGAT, local	JavaGAT, sftp	JavaGAT, smb	jcifs.smb
1)	1652	85522	9881	3061
2)	1573	70829	10697	3173
3)	1525	85012	10221	3144
average	1583	80454	10266	3126
MB/sec	63	1	9	31

Figure 6.3: The results of the input stream measurements with 100MB file size

implementations is not significant compared to the case of 50 MB: JavaGAT 9 MB/sec and `jcifs.smb` 31 MB/sec.

The results of the copy from input stream of `java.io` to output stream of `jcifs.smb` and `org.gridlab.gat.io` with file size 100 MB is shown in the figure 6.4. The speed difference of JavaGAT and `jcifs.smb` implementations are almost the same as in the 50 MB case: JavaGAT has 10 MB/sec speed and `jcifs.smb` 24 MB/sec.

	JavaGAT, local	JavaGAT, sftp	JavaGAT, smb	jcifs.smb
1)	1723	65659	9587	4090
2)	5731	57781	9539	4099
3)	1694	63485	9649	3926
average	3049	62308	9591	4038
MB/sec	32	1	10	24

Figure 6.4: The results of the output stream measurements with 100MB file size

The second benchmark is file copy. Two file copies use the copy method of JavaGAT `File` class. The third one uses the `copyTo` method of `jcifs.smb.SmbFile` class. The file copy is done on a single computer. The file copy is a samba to samba file transfer or only a local copy. JavaGAT samba adaptor is compared to the local file copy with JavaGAT and to native java implemented samba file copy (`jcifs.smb`). Other JavaGAT adaptors do not enable the file copy on the same machine. We have measured the speed of file copy with file size of 50 MB and 100 MB.

The results of file copy with file size 50 MB is represented in figure 6.5 and with file size of 100 MB in figure 6.6. In the first row we can see which implementation is used for the file copy: local is the local file adaptor and the smb is samba file adaptor of JavaGAT, and `jcifs.smb` is the fourth column. In the following three rows the result of the measurements are shown in milliseconds then the average values. In the last row we can see the speed in MB/sec.

In figure 6.5 we can see there is noticable difference between the local and

the samba file copy. The local file copy is more than 10 times faster (local: 53 MB/sec, samba: 5 MB/sec).

	local	smb	jcifs.smb
1)	912	9323	9254
2)	954	9102	8949
3)	938	9438	9139
average	934	9287	9114
MB/sec	53	5	5

Figure 6.5: The results of the file copy measurements with file size 50MB

In figure 6.6 we can see, if copy is 100 MB of data, then the speed difference is lower (local: 51 MB/sec and samba: 8 MB/sec).

	local	smb	jcifs.smb
1)	1961	11332	11144
2)	1857	11466	11277
3)	1991	11414	11191
average	1936	11404	11204
MB/sec	51	8	8

Figure 6.6: The results of the file copy measurements with file size 100MB

6.7 Conclusion

The speed of JavaGAT and `jcifs.smb` implementations can be the most relevant for us. But we can see the speed of the local and sftp file adaptors in the stream benchmarks, as well. The local file adaptor is the faster (32-63 MB/sec), and the sftp adaptor is the slowest (1-5 MB/sec). This can be explained by the underlying protocols. Local file copy uses the functions of the operating system, which should be fast for local file copy. Sftp uses more complicated authentication and security functions, and it is designed for secure network file transfers. The differences between JavaGAT and `jcifs.smb` can show us the overhead of JavaGAT. JavaGAT is slower with 13-23 MB/sec. The speed of samba input/output stream adaptors in our benchmarks: 5-10 MB/sec.

The file copies have no noticable differences between JavaGAT and `jcifs.smb` implementations. The speed differences compared to the local file copy are 48 MB/sec and 43 MB/sec, and the speed of the samba file copies is: 5-8 MB/sec.

Chapter 7

Designing and creating the Globus 4 adaptors

In this chapter we discuss the GT4 adaptors for JavaGAT. First we take a look at the programming and testing environment. Then the adaptors are represented. The usage and functions of the adaptor are explained in a following section. In the end of the chapter the performance benchmarks are shown.

7.1 DAS-3

Both programming and running environment are set up on the DAS-3[1] supercomputer. For that reason the hardware and software details of DAS-3 come here.

DAS-3 stands for Distributed ASCI Supercomputer 3. DAS-3 is a wide area distributed system designed by the ASCI, Advanced School for Computing and Imaging. The DAS-3 connects five cluster with hybrid optical networks. It intends to support researchers who work on aspects of parallel, distributed and grid computing and large-scale multimedia content analysis. Besides the Vrije Universiteit four other organizations operate clusters (Leiden University, University of Amsterdam, Delft University of Technology, The MultimediaN Consortium).

The DAS-3 cluster at the Vrije Universiteit in Amsterdam consists of 85 dual-CPU / dual-core 2.4 GHz AMD Opteron DP 280 compute nodes, each having 4 GB of memory and 250 GB of local HD space. The cluster head node also consists of a dual-CPU / dual-core 2.4 GHz AMD Opteron DP 280, but with 8 GB of memory and an additional RAID6 storage system of 10 TB. The cluster is equipped with 1 and 10 Gigabit/s Ethernet, as well as a high speed Myri-10G interconnect. The nodes of the other cluster sites have similar parameters to the nodes at Vrije Universiteit. If we put everything together, the DAS-3 has 272 computing nodes.

The operating system the DAS-3 runs is Scientific Linux. The Globus Toolkit 4.0.3 version is installed. For more details on software configuration see appendix C.1.

[1] http://www.cs.vu.nl/das3/index.shtml

7.2 Globus 4 adaptors

The logical view of the adaptors is represented in Figure 7.1. The grid application is built on the top of the JavaGAT. The children of the JavaGAT show which function of JavaGAT is implemented by the adaptors: I/O or resources. On the next level of the tree we can see the API used by the adaptor: CoG refers to the JavaCoG abstractions and WS and pre-WS means the Web Service and pre-Web Service Globus APIs. In the leaves the names of the different adaptors are shown. Our Globus 4 binding for JavaGAT have four adaptors.

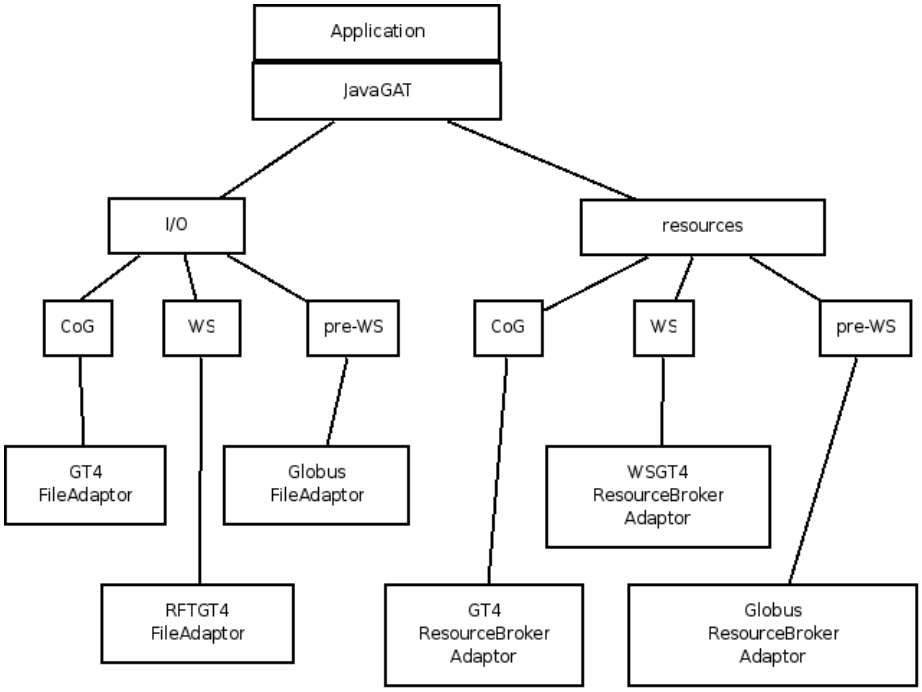

Figure 7.1: Adaptors in JavaGAT

Two different implementations of File adaptors and other two implementations of Resource adaptors are done, the adaptors are referred with these names in the later discussion:

GT4FileAdaptor The GT4FileAdaptor is implemented with the JavaCoG kit abstractions. It supports every JavaGAT file operation except `createNewFile` , `isHidden` operations. File copy works between local and GridFTP file systems and between GridFTP sites supporting third party copy.

RFTGT4FileAdaptor The RFTGT4FileAdaptor is implemented with Globus API, using the WS based services. This adaptor only supports the file coping and the file deletion functions of JavaGAT. The file copy can be done between Globus grid sites. The file deletion is also available only on the Globus file system.

GT4ResourceBrokerAdaptor The GT4ResourceBrokerAdaptor is implemented with JavaCoG abstractions. It provides the job submission support of

JavaGAT.

WSGT4ResourceBrokerAdaptor The WSGT4ResourceBrokerAdaptor is implemented with Globus API, using the WS based services. It covers the job submission support for JavaGAT.

7.3 Implementation of the Globus 4 adaptors

7.3.1 Introduction

In this section we can see a comprehensive disscussion of the implementation details with some important remarks. The mentioned parts of the source code are included in the appendix.

7.3.2 GT4FileAdaptor

In the `GT4FileAdaptor` the JavaCoG abstractions are used. The `GT4FileAdaptor` is an abstract class. The constructor and the copy function of that is overridden by the subclasses (`GT4GridFTPFileAdaptor` and `GT4LocalFileAdaptor`) and they pass a provider `String` to the superclass. The `copy` method is also implemented by the subclasses. The value of that `String` could be `gsiftp` or `local`.

The following methods of the `FileCpi` class are realized: copy, canRead, canWrite, delete, exists, getAbsoluteFile, getAbsolutePath, isDirectory, isFile, isHidden, lastModified, length, list, mkdir, setLastModified, setReadOnly. Every function wraps the appropriate method of the `FileResource` object. The only exception is the file copy. The `FileResource` object is initialized in the constructor, the following parameters have to be passed to it:

- String of the provider: this value comes from the subclasses. Now we have two subclasses which support gsiftp or local protocols.
- `SecurityContext`: to create a `SecurityContext` object we need to get a valid credential. This credential is acquired to use `GlobusSecurityUtils.getGlobusCredential` method which is implemented with the earlier version of Globus adaptor. The credential with `null` value means that the default credential is going to be used.
- `ServiceContact`: the creation of the `ServiceContact` requires a service contact string: this string is produced from the file location.

Implementation of the `copy` method in the `GT4GridFTPFileAdaptor` class:

- First we handle the case that the source is a directory. If it is a directory the `copyDirectory` method is called, which method is implemented in the `FileCpi` class and does recursive copy of a directory, using the abstract methods of the `FileCpi` class.
- In the next step the case is checked when the source is file and the destination is a directory. To get working that the filename should be passed after the destination directory. Otherwise the Globus tries to overwrite the destination and the copy will fail.

- Then the destination is tested if it is a local file or not. If it is a local file we can use the `getFile` method of the `FileResource` class.

- In the following step if the scheme of the destination is "any" then the `copyThirdParty` method is called with all the possible providers, until it succeeds to copy. If the scheme of destination is known, the `copyThirdParty` method is called with the appropriate provider.

- The `copyThirdParty` method copies a file using the task model of the JavaCoG kit abstractions. If the copy is failed, the method throws an exception.

The other methods use the appropriate methods of the `FileResource` or the `GridFile` classes. We can get a `GridFile` object by invoking `getGridFile` method of the `FileResource` class.

7.3.3 RFTGT4FileAdaptor

The `RFTGT4FileAdaptor` is implemented with web services of the Globus 4 API. That adaptor only covers the file copy and file delete functions, because the Reliable File Transfer (RFT) service supports only these operations in Globus 4.

The `RFTGT4FileAdaptor` consists of two classes: `RFTGT4FileAdaptor` and the `RFTGT4NotifyCallback`. The `RFTGT4FileAdaptor` extends the `FileCpi` class, while the `RFTGT4NotifyCallback` implements the `NotifyCallback` interface of the Globus API.

The `RFTGT4NotifyCallback` class takes the responsibility for receiving messages from a notification service through its `deliver` method. The state of the file transfers is included in the messages. After receiving it the state is converted to an `OverallStatus` object, and passed to the `RFTGT4FileAdaptor`.

The key steps of a file transfer or a file deletion:

- First a `ReliableFileTransferFactoryPortType` object is created from the URL of the file.

- Then the credential end points are fetched and the credentials are delegated.

- The file transfer (`TransferType`, `RFTOptionsType` and `TransferRequestType` classes are initialized) or the deletion (`DeleteType`, `DeleteRequestType`) is set up.

- Then an `EndpointReferenceType` is created, the security and operation properties are passed. Termination time is set to 20 minutes both in the endpoint reference and in request. Then we subscribe for notification. The topic of notification is the overall status of the RFT service: `RFTConstants.OVERALL_STATUS_RESOURCE`.

- In the notification messages we get the numbers of active, finished, cancelled, failed, pending and restarted operations. Our file transfer or deletion operation waits until there are no active, pending or restarted task.

7.3.4 GT4ResourceBrokerAdaptor

`GT4ResourceBrokerAdaptor` is implemented with abstractions of the JavaCoG kit. The resource broker adaptor can submit and manage jobs. Our resource broker uses `Sandbox`. `Sandbox` handles the file staging and clean up. The implementation of `Sandbox` uses the JavaGAT classes to carry out the file operations. `GT4ResourceBrokerAdaptor` has three classes: `GT4ResourceBrokerAdaptor`, `GT4Job` and `GT4StatusListener`. The steps of the job submission are the following:

- The `submitJob` method of the `GT4ResourceBrokerAdaptor` creates and returns a `GT4Job`. The `Sandbox`, the `JobSpecification` and the `Service` objects are passed to the `GT4Job`. `Sandbox` is initialized by the resource broker. `JobSpecification` object is created from the `JobDescription` object, which is a GAT object, and contains the data of the job. The `JobSpecification` holds the location of the executable file, input, output and error files, the arguments for the executable file and the environment variables. The location of the standard input, output and error files are stored in the `Sandbox` object. The `Service` object contains information about the Globus 4 job submission service. It is initialized with the appropriate security context and with the hostname. The hostname is the contact string of the job submission node and it is taken from the `JobDescription`, which is set up by the application developer.

- After the `GT4Job` gets these objects, a `Task` object of the JavaCoG abstractions is created with the `JobSpecification` and with the `Service`.

- Then a GT4StatusListener is created, which handles the notification messages, and changes the state of the `GT4Job`. The possible states of a job are the not same in JavaGAT and JavaCoG, the assignments between the states are shown in the appendix TBD.

- Finally, we create an `ExecutionTaskHandler` object and it submits the job.

7.3.5 WSGT4ResourceBrokerAdaptor

The `WSGT4ResourceBrokerAdaptor` uses the web services of the Globus 4 API. That implementation also relies on the JavaGAT `Sandbox` object. The adaptor is built up of three classes: `WSGT4ResourceBrokerAdaptor`, `WSGT4Job` and `WSGT4NotificationCallback`.

The steps of the job submission are the following:

- The `WSGT4ResourceBrokerAdaptor` creates the `Sandbox`, `JobDescriptionType` and gets the credential. The `JobDescriptionType` object is created from an XML `String`. That XML string contains the data of job: the path to the executable file, the path of the standard input, output and error, arguments and environment variables. The standard input, output and error files are local path in the `Sandbox`.

- Then we create the `WSGT4Job` object.

- The `WSGT4Job` creates first the job factory service stub: `factoryEndpoint` object of an `EndpointReferenceType` class. Then the delegation endpoints are fetched and the credentials are delegated. The credentials might be

used by the RFT as well, because it can contact the GridFTP server. But to create delegated credentials for RFT is not necessary for us, because the adaptor uses `Sandbox`.

- Next step is the creation of the job resource. A `CreateManagedJobInputType` object is created. This object needs an identity, termination time, and the job description.
- After that, we subscribe for notifications of job status. So the topic of notification is set to `ManagedJobConstants.RP_STATE`. The `deliver` method of the `WSGT4NotifyCallback` class receives the notifications after the job is submitted.
- Then we get the endpoint of the job. The job is submitted.
- The notifications are received by the `WSGT4NotifyCallback`. This object extracts the state or the fault. Then it passes to the `WSGT4Job` and sets the state of it.

7.3.6 The building and running environment

The JavaCoG kit is placed under the JavaGAT location to in own directory (named `cog-4_1_4`). The Globus API libraries are added to the classpath by modifying the `build.xml` files.

The running scripts are also modified, the classpath includes JavaCoG and Globus locations. More than one running script is created because of the incompatible version of the UUIDGenFactory class (which exists both in the JavaCoG and Globus libraries). The `client-config.wsdd` file has be copied to the current directory from the `$GLOBUS_LOCATION`, otherwise the WS based adaptors fail (the reason is unknown).

7.4 Comparison of the adaptors

In this section we first discuss the difference between the file adaptors. Then the resource adaptors are compared.

The main differences between the `GT4FileAdaptor` and the `WSGT4FileAdaptor` implementations are in the functions. Here we can see the features of both adaptors, first comes the `GT4FileAdaptor`:

- Almost every method of the `FileCpi` is implemented: `copy`, `canRead`, `canWrite` , `delete`, `exists`, `getAbsoluteFile`, `getAbsolutePath`, `isDirectory`, `isFile`, `lastModified`, `length`, `list`, `mkdir` and `setReadOnly`. The other file operations are not implemented because GridFTP does not support them.
- The file copy works between the local file system and GridFTP, and between GridFTP sites. Besides of this other protocols are supported by the JavaCoG kit, so by extending the `GT4FileAdaptor` class we can support ssh, webdav or condor file access protocols.

The only operation supported by `RFTGT4FileAdaptor` is the file copy and deletion. The file copy can be done only between Globus 4 sites.

The resource broker adaptors cover the operations. We can use them to send a single job to a Globus 4 grid. The staging and cleaning up of files are done by the `Sandbox`, which can use any JavaGAT file adaptors to carry out these operations.

7.5 Tests

There are two small applications for testing: `TestFileAdaptor` and `TestResourceBroker` . The command line invocation of them is described in the appendix C.2.

7.6 Measurements

The test applications have been used for benchmarks. The speed of the file copy has been measured with the `TestFileAdaptor`. The time of copy method is measured by checking the time before and after the invocation of the `copy` method.

First the file copy benchmarks are represented. Two measurements have been done. The first one of the file copy is a copy with file of 1MB data, and the other one with 100MB file size. The file copies are between two sites: fs2.das3.science.uva.nl and fs3.das3.tudelft.nl, the only exception is the local file copy that uses only the fs0.das3.cs.vu.nl file server. The results are represented in Figures 7.2 and 7.3. In the first column we can see which adaptor has been used for the file copy. The next three columns show the result of the benchmarks in milliseconds, which are followed by the average values. In the last column of the table 7.2 the speed of the copy is represented in KB/sec and in the table 7.3 in MB/sec, as well.

	1)	2)	3)	average	KB/sec
`GT4GridFTPFileAdaptor`	2751	2813	2794	2786	367
`RFTGT4FileAdaptor`	12880	14105	20645	15876	64
`GridFTPFileAdaptor`	2394	2359	2383	2378	430
`LocalFileAdaptor`	21	21	29	23	44521

Figure 7.2: File copy of 1MB data

	1)	2)	3)	average	MB/sec
`GT4GridFTPFileAdaptor`	5062	4789	4470	4773	20
`RFTGT4FileAdaptor`	18540	23366	18105	16670	5
`GridFTPFileAdaptor`	4935	3892	8369	5732	17
`LocalFileAdaptor`	1090	1136	1146	1124	88

Figure 7.3: File copy of 100MB data

As we can see on the figures 7.2 and 7.3 every implementation has some overhead of file copy compared to a local file copy. The `RFTGT4FileAdaptor` is the slowest in both cases with 15876 ms and 16670 ms. The other two Globus adaptors have similar speed. When 1MB is copied the `GridFTPFileAdaptor` is

the faster with 408 ms. In the other case the `GT4GridFTPFileAdaptor` has better performance, the difference is 959 ms.

The speed of the job submission has been evaluated with the `TestResourceBroker` application. For the resource broker the speed of simple job submission is benchmarked. The job is `/bin/false` command without any file staging. The time is checked before we call the `submitJob` method and after the job has been done. The job is a simple command (`/bin/false`) without any input, output or file staging. The job submission application (`TestResourceBroker`) is invoked from the fs0.das3.cs.vu.nl and the job is submitted to the fs2.das3.science.uva.nl site. The `LocalFileAdaptor` uses the fs0.das3.cs.vu.nl host. We measured the benchmarks ten times, the result are represented int the figure 7.4.

	GT4 ResourceBroker Adaptor	WSGT4 ResourceBroker Adaptor	Globus ResourceBroker Adaptor	Local ResourceBroker Adaptor
1)	9545	25886	13272	1458
2)	25669	9842	13266	1531
3)	21644	16992	13364	1477
4)	19237	23691	13402	1464
5)	19767	24370	13312	1462
6)	23931	23540	13433	1464
7)	25049	26432	13326	1461
8)	21456	9231	13299	1464
9)	25853	24893	13243	1465
10)	21795	21401	13322	1497
avg	21394	20627	13323	1474

Figure 7.4: Simple job submission measurements result

The bottom of table shows the average job submission times for each adaptor. The average time of the job submissions shows us that the `GT4ResourceBroker` and `WSGT4ResourceBroker` have almost the same speed, the time difference is only 767 ms, and the `WSGT4ResourceBroker` is more efficient with 20627 ms. The `GlobusResourceBroker` has significally better performance with 13323 ms. Because of obvious reasons the local file adaptor has the lowest overhead, the average time of a job submission is 1474 ms.

Chapter 8

Conclusions

JavaGAT adaptors already have already been created for Globus grid. However, the Globus API and the related toolkits and software frameworks have been influenced by the new release of Globus Toolkit. The implementation of the Globus 4 services is based on Web Services. For that reason Globus 4 API has also changed, and new implementations of adaptors have become possible. But the continual evolution of the grid middleware is an obstacle for the rapid grid application development. That is why it is very important the improvment of the higher level toolkits. JavaCoG offers two abstract level interfaces to Globus. The jGlobus library allows the developer to create Globus applications independently from the version, while the CoG abstractions based applications can be grid middleware independent. There are Globus 4 adaptors for JavaGAT which use jGlobus API, so we decided to use Globus 4 API and CoG abstractions to access the Globus middleware services from JavaGAT. Two sorts of adaptors represented in this document: file and resource broker adaptors.

The implemented file adaptor has remarkable differences in the functions. The file operations using Web Services are restricted for file copy and deletion. Therefore the `RFTGT4FileAdaptor` only supports these operations. The performance of that adaptor is also weak compared to the `GridFTPFileAdaptor` it is only 64 KB/sec (file size 1MB) or 5 MB/sec (file size 100MB). This low performance can be explained with the overhead of the Web Services. The implementation of that adaptor is as well more complicated because it uses low level API.

The operations of `GT4FileAdaptor` are almost complete. The implementation is also efficient, because if we look at the file copy measurements the performance of the `GridFTPFileAdaptor` adaptor is approximately the same:

- GT4GridFTPFileAdaptor: 367 KB/sec (file size 1MB) and 20 MB/sec (file size 100MB).
- GridFTPFileAdaptor: 430 KB/sec (file size 1MB) and MB/sec (file size 100MB).

For the programmers the CoG abstractions offers clear and understandable programming model.

The resource broker adaptors know the same operations. Both of the `WSGT4ResourceBrokerAdaptor` and `GT4ResourceBrokerAdaptor` have considerable lower performance than the `GlobusResourceBroker`, it is 65% (WS implemented) and 62% (CoG abstractions) of the `GlobusResourceBroker`.

Appendix A

JavaCoG Kit

A.1 JavaCoG providers

The possible values for JavaCoG provider are

`gt2ft, gsiftp, condor, ssh, gt4ft, local, gt4, gsiftp-old, gt3.2.1, gt2, ftp, webdav.`

JavaCoG provider aliases:

```
webdav <-> http;
local <-> file;
gsiftp-old <-> gridftp-old;
gsiftp <-> gridftp;
gt4 <-> gt3.9.5, gt4.0.2, gt4.0.1, gt4.0.0
```

Appendix B

Samba adaptors

B.1 Enviroment

The branch of the JavaGAT was downloaded with `svn` from here: `https://gforge.cs.vu.nl/svn/javagat/branches/Balazs`. The `$GAT_LOCATION` shell variable is set to to the JavaGAT root directory. The Java version 1.5 should be installed and `$JAVA_HOME` set.

B.2 Usage of the test program for the Samba adaptor

The applications can be invoked from the command line. For example if the actual directory is the `$GAT_LOCATION`:

```
./bin/run_gat_app myprobe.SmbFile [parameters]
```

The usage of the applications: `SmbFile`:

```
Usage: command location [username password]
command: list, listFiles, exists, isDirectory,
length, mkdir, mkdirs, delete, canRead, canWrite,
createNewFile, isFile, isHidden, lastModified
```

`SmbRAFile`:

```
Usage: location [username password]
```

`SmbInputStream`, `SmbOutputStream`:

```
Usage: location [username password]
```

B.3 The testing machine

The testing machine is Pentium IV laptop @1600MHz, with 1024MB ram, 5400 rpm hard disk, installed Ubuntu linux, Samba and JavaGAT.

B.4 Measurments for Samba

Command line invocation of the benchmarks:

- copy local file using JavaGAT sftp adaptor (local → sftp)

```
./bin/run_gat_app myprobe/CpLocalToGAT /path/file \
  sftp://host//path/file user password
```

- copy local file using JavaGAT smb adaptor (local → gat-smb)

```
./bin/run_gat_app myprobe/CpLocalToGAT /path/file \
  smb://host/share/path/file user password
```

- copy local file using jcifs smb FileOutputStream (local → jcifs-smb)

```
./bin/run_gat_app myprobe/CpLocalToGAT /path/file \
  smb://host/share/path/file user password
```

- copy local file using JavaGAT localfile adaptor (local → gat-local)

```
./bin/run_gat_app myprobe/CpLocalToGAT /path/file \
  /path/file
```

Appendix C

Globus 4 adaptors

C.1 DAS-3 software environment

These softwares are used to deal with JavaGAT:

- Globus Toolkit, version 4.0.3
- Java, version 1.5.0_07
- Apache Ant, version 1.6.5
- svn, version 1.1.4

C.2 Test applications

The test applications can be invoked from command line with the appropriate environment settings. The parameters for the `TestFileAdaptor` and for the `TestResourceBroker` are the following:

```
Usage: TestFileAdaptor command [argument] fileuri
The fileuri is URI of a file. The commands are
the next, some of these need argument. Theprefkey and prefvalue parameters can be usedseveral t
  --delete      deletes file
  --prefkey key         set preference key for GAT
  --prefvalue value     set preference value for GAT
  --createfile  creates a file
  --exists      checks the file is exists or not
  --absolutepath        returns the absolute path
  --isdir       checks the file for directory
  --lastmod     returns the time of last modification in milliseconds
  --setlastmod ms       sets the lastmodification to ms given in milliseconds
  --size        returns the size of the file
  --list        lists the directory
  --mkdir       creates a directory
  --setreadoly  sets the file read-only
  --copy fileuri        copies file to destination URI
  --canwrite    checks file for writing
  --canread     checks file for reading
```

```
 Usage: TestResourceBroker [perferences] \
 --exe executable \
 [--input inputfile] \
 [--output outputfile] \
 [--error errotoutputfile \
 contact
The contact is like that:
[protocol://]{hostname|hostaddr}[:port][/service]).
The preferences can passed like this:
--perfkey key --perfvalue value
```

References

- Rob V. van Nieuwpoort: GAT Tutorial,
  `https://gforge.cs.vu.nl/projects/javagat/`
- Borja Sotomayor: The Globus Toolkit 4 Programmer's Tutorial,
  `http://gdp.globus.org/gt4-tutorial/multiplehtml/index.html`
- Gregor von Laszewski and Kaizar Amin: Java CoG Kit Abstraction Guide,
  `http://wiki.cogkit.org/index.php/Java_CoG_Kit_Abstraction_Guide`