HEP Community Grid

Stage One Report on

WP 3: Distributed Data Analysis under usage of Grid Resources – Gap Analysis

GSI Darmstadt, 11.11.2006
Version: 1.32

Table of Content

1 Abstract

TODO: Write this part as the last one

At GSI distributed analysis tools under usage of grid resources are being developed within work package 3 of the HEP community Grid.
A starting point is the analysis framework ROOT. Making use of a set of abstract classes provided by ROOT (*TGrid* ...) an interface to gLite is being created to enable Grid access directly from within ROOT. This includes querying the File Catalogue, job submission, getting job status and output. By combining several stand-alone PROOF based analysis facilities using existing Grid Middleware large dynamically generated Grid Analysis Clusters can be created.

2 Introduction

TODO: Write this part as the last one

3 Distributed parallel data analysis

The aim of distributed parallel data analysis on the Grid is to enable physics collaborations and individual physicists to use the power and resources of the Grid.

The four experiments ATLAS, ALICE, CMS, and LHCb, at CERN will collect, in the first year, an amount of about 20 Petabytes of data. Also in the following years the LHC (Large Hadron Collider) experiments will produce data in the order of Petabytes, so it requires a lot of storage space as well as CPU power in order to perform simulations and to do the analysis of these data (see Figure 1).

Summary of Computing Capacity Required for all LHC Experiments in 2007

source: CERN/LHCC/2001-004 - Report of the LHC Computing Review - 20 February 2001

(ATLAS with 270Hz trigger)

	CERN			Regional	Grand
	Tier 0	Tier 1	Total	Centres	Total
Processing (K SI95)	1,727	832	2,559	4,974	7,533
Disk (PB)	1.2	1.2	2.4	8.7	11.1
Magnetic tape (PB)	16.3	1.2	17.6	20.3	37.9

1 SPECint95 = 10 CERNunits = 40 MIPS
today dual PIII ~ 100 SPECint95

Figure 1: Summary of computing capacity of LHC experiments [1]

Conventional Mass Storage Systems and Batch Systems of individual computing centers can handle very well subsets of this dataset. When scaling up to the total amount of data one requires an equivalent scale change from single computing centers to a complete Grid of computing centers.
Each LHC experiment will involve an average of 1000 physicists. This is an important parameter since all collaborators must have access to the data. In this case Grid can be the only solution for coordinated efficient production, data distribution, and data analysis.
In this paper we will discuss not only advantages, which a production Grid would bring to the HEP (High Energy Physics) community, but mostly possible ways to improve the speed of data analysis, to make it more convenient for users. The main idea is how to define a general way of distributed data analysis on the Grid for all experiments in the HEP community.
Today, the Grid gives us the possibility to access resources of different sites and to use it as if it would be one local batch farm with one single storage system. The linking of geographically dispersed computer systems can lead to staggering gains in computing power, speed, and productivity. The Grid can solve larger, more complex problems in a shorter time, makes better use of modern hardware, and eases collaborations among distributed organizations.
But so far there is no general standard defined which can help physicists to make their job more efficient by using the Grid. Each of the LHC collaborations is forced to use its own home-made schema or framework to put the Grid on their needs.

One can summarize and list the following problems:

- There is no general and Grid enabled data analysis schema.
- There is no standardized and generally accepted API of the Grid; we are talking here about a high level API, which could be generally used by an end-user to satisfy his/her needs in parallel data processing on the Grid in the simplest possible way.

Our task is to generalize the capabilities and to investigate the current Grid market as well as the possibility for creating a model, based on which the HEP community could get one generic way for an efficient and user-friendly Grid use. We would like to find the model, which could be most easily adapted, which would be flexible, and would not require a special knowledge from the user. Our work must define this model and also it should define criteria of success for this model. We have to create an operational solution, which is a very important part of the model's implementation.

3.1 The ALICE model of distributed and parallel analysis

After a short pre-stage strategy investigation, we decided to use in the role of a starting point for our project one existing and operational distributed analysis model, which has been implemented by the ALICE collaboration and is so-far working only for the ALICE experiment, because of its restrictions. But we think that this model has a lot of potential and could be populated and spread to a wider range of experiments and Grids.

The ALICE model involves AliROOT (the ALICE analysis and simulation framework, which is completely based on ROOT [4]), AliEn (Grid middleware developed by the ALICE collaboration) [5] and PROOF (The Parallel ROOT Facility) [6].

ROOT is obviously becoming one of the most popular physics analysis toolkits. It is a de facto standard framework in the HEP community as well.
Shortly saying it provides three key points:

- the possibility to do interactive analysis work in familiar C++ style syntax;
- data visualization,
- an object-oriented I/O system.

It is successfully used within frameworks of different experiments as an "all in one" solution.

PROOF extends a workstation based concept of ROOT to a parallel processing on a cluster (see Figure 2), where:

- user procedures are kept identical during an analysis session,
- tasks are distributed automatically in the background.

The motto of PROOF is: "**Bring the KB to the PB, not the PB to the KB!**"

AliEn (**ALI**CE production distributed **En**vironment) as a Grid platform for distributed production and analysis provides two key elements:

- a global file system -- files are indexed and tagged in a virtual file catalogue and are globally accessible from everywhere;

- a global queue system, and global job scheduling according to resource requirements.

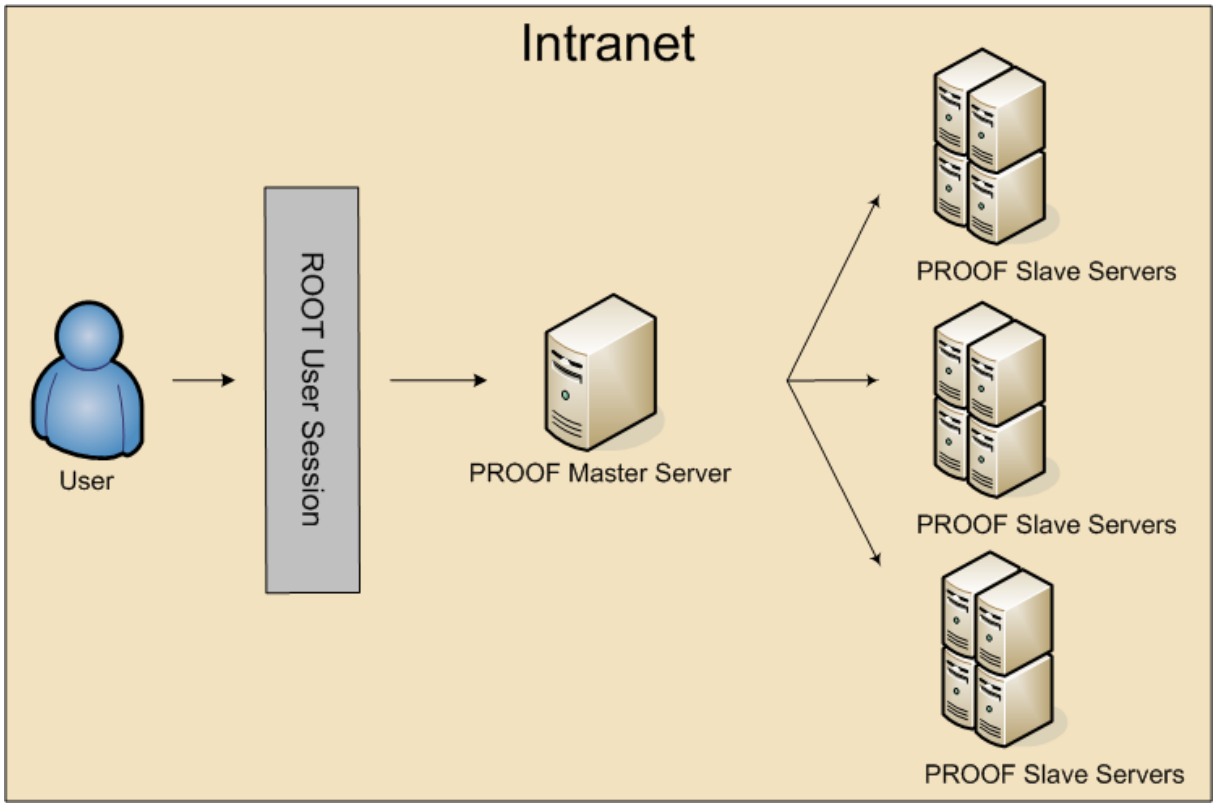


Figure 2: Local PROOF cluster

3.1.1 Batch Interactive Data Analysis with AliEn and ROOT

Normally in order to process an analysis using AliEn and ROOT one should use a ROOT AliEn plug-in, which is currently provided by the standard ROOT installation package and AliEn developers. The plug-in is an implementation of a ROOT Grid Interface. The ROOT Grid Interface is a subset of abstract C++ classes, which are defining Grid interface operations for ROOT users. In Figure 3 we present an UML class diagram of the ROOT Grid Interface is shown.
The interface consists of the following classes:

- ***TGrid*** (TGrid.h(cxx)): A pure abstract base class, which defines the interface to common GRID services.
- ***TGridJDL*** (TGridJDL.h(cxx)): A pure abstract class. It is used to generate JDL files for job submission to the Grid.
- ***TGridJob*** (TGridJob.h(cxx)): A pure abstract class. The class defines an interface to a GRID job.
- ***TGridJobStatus*** (TGridJobStatus.h(cxx)): A pure abstract class. The class contains the status of a Grid job.
- ***TGridResult*** (TGridResult.h(cxx)): A pure abstract base class, which defines an interface to a GRID result. Objects of this class are created by TGrid methods.

- ***TGridCollection*** (TGridCollection.h): a class which manages files collections on the Grid.

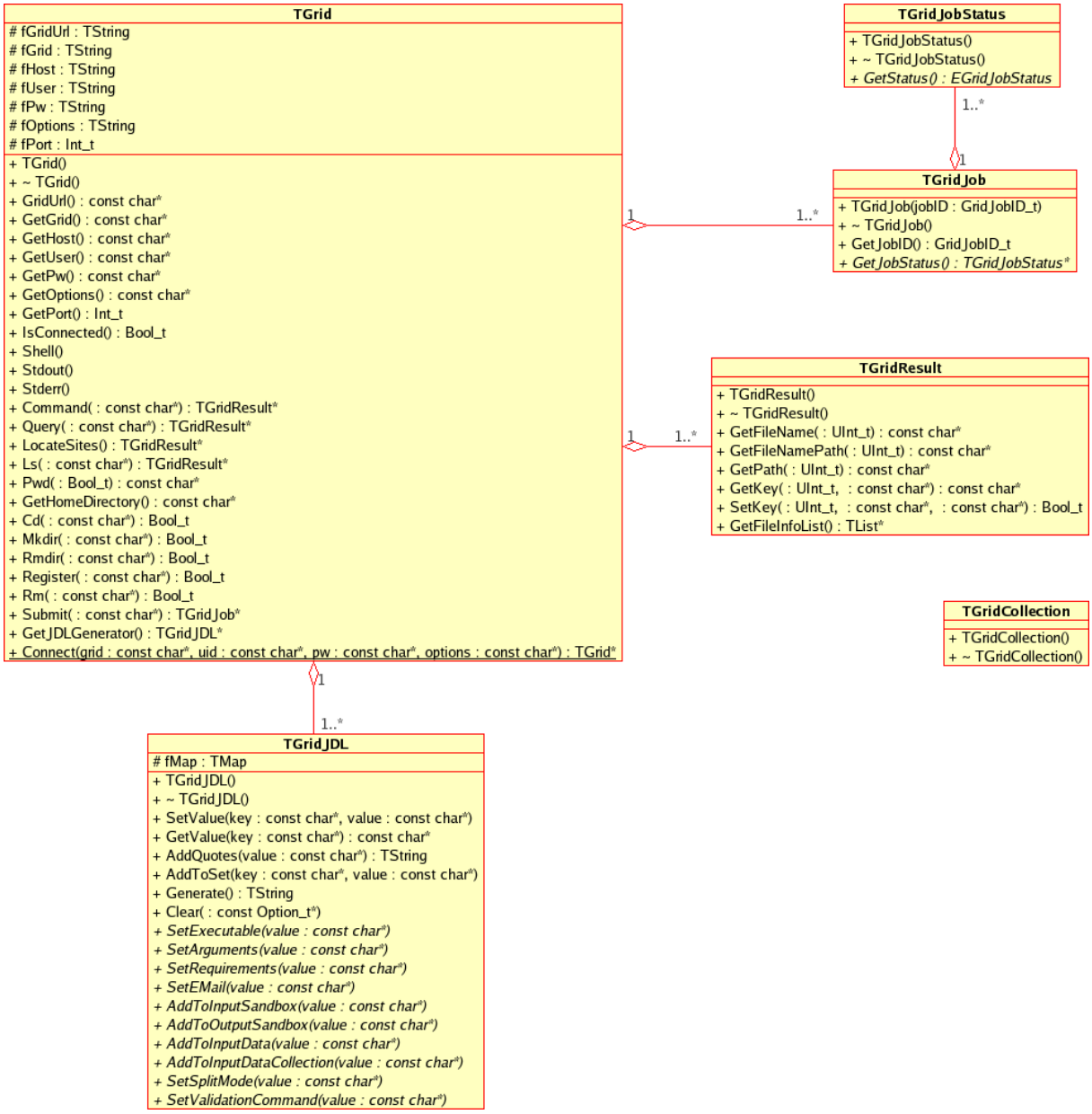


Figure 3: UML class diagram of the ROOT Grid Interface

In Figure 4 an UML class diagram of the ROOT plug-in for the AliEn middleware is shown. This plug-in is the first concrete implementation of a ROOT Grid interface, which is already operational for AliEn users.

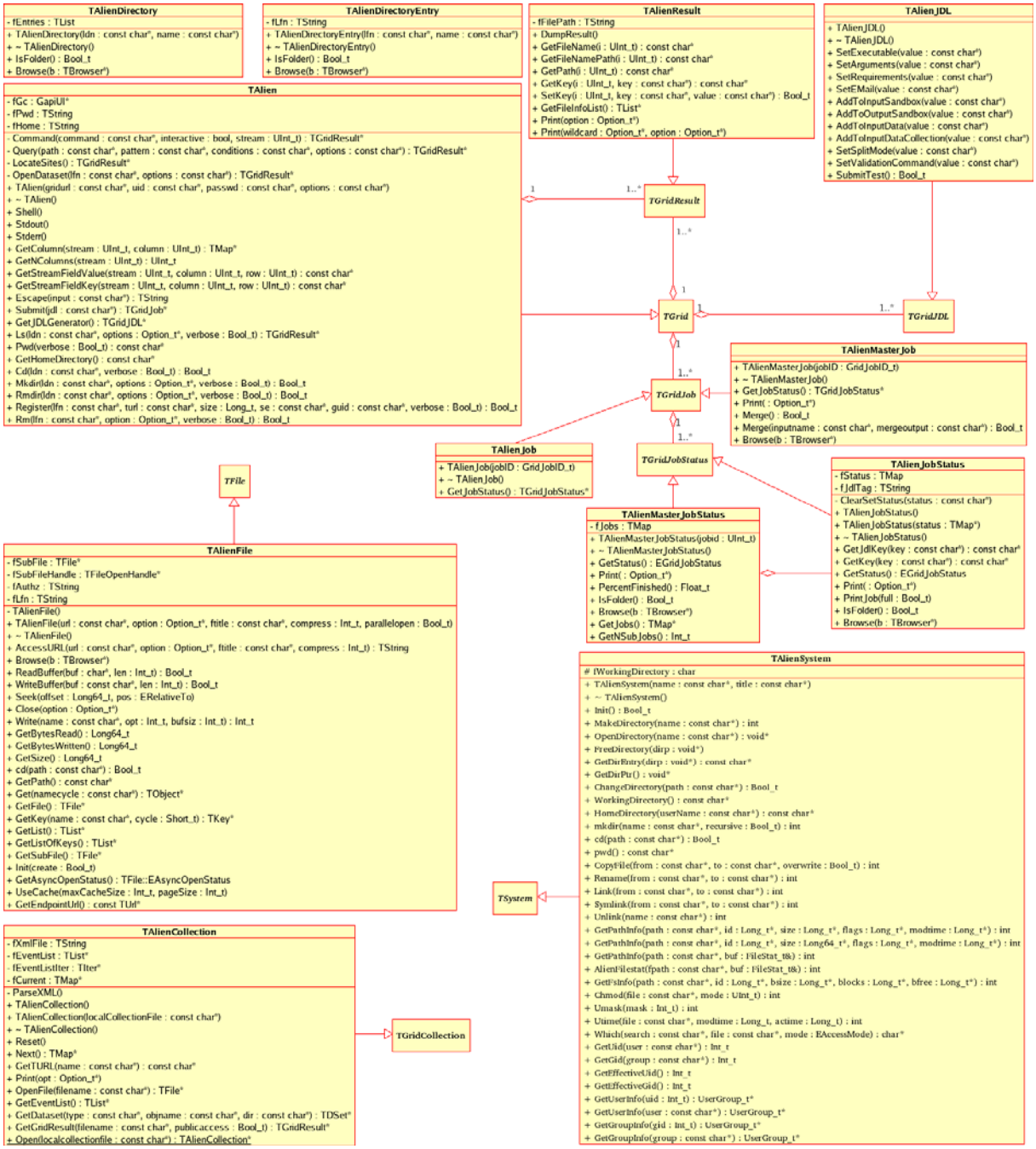


Figure 4: UML class diagram of the ROOT plug-in for the AliEn middleware

The analysis process, which based on the AliEn plug-in, consists of the following steps, which could be processed in the ROOT C++ interpreter console (manual enter mode), by a ROOT macro or by using a C++ program with enabled ROOT support:

- Query for input data in the AliEn File Catalogue;
- registration of the input data in ROOT;
- creation of the corresponding JDL file;
- submission of a single job request, which is spawned afterwards by the AliEn engine;

- status Check;
- merging of interactive result files;
- merging of batch result files.

3.1.2 Interactive Data Analysis with AliEn and PROOF

The next step is to marry PROOF, which has been basically designed for doing data analysis on local batch farms in parallel, and AliEn.
There are several basic requirements for Interactive Data Analysis made with the help of AliEn and PROOF:

1. The data to be analyzed have to be stored in ROOT files (***TTrees*** or ***TKeys***).
2. PROOF servers have to load shared libraries for user and experiment code.
3. Analysis code has to be inserted into the automatically generated selector macro for the objects to be analyzed: ***obj->MakeSelector();***

A schema of a PROOF based interactive analysis using AliEn Grid, is shown in Figure 5.

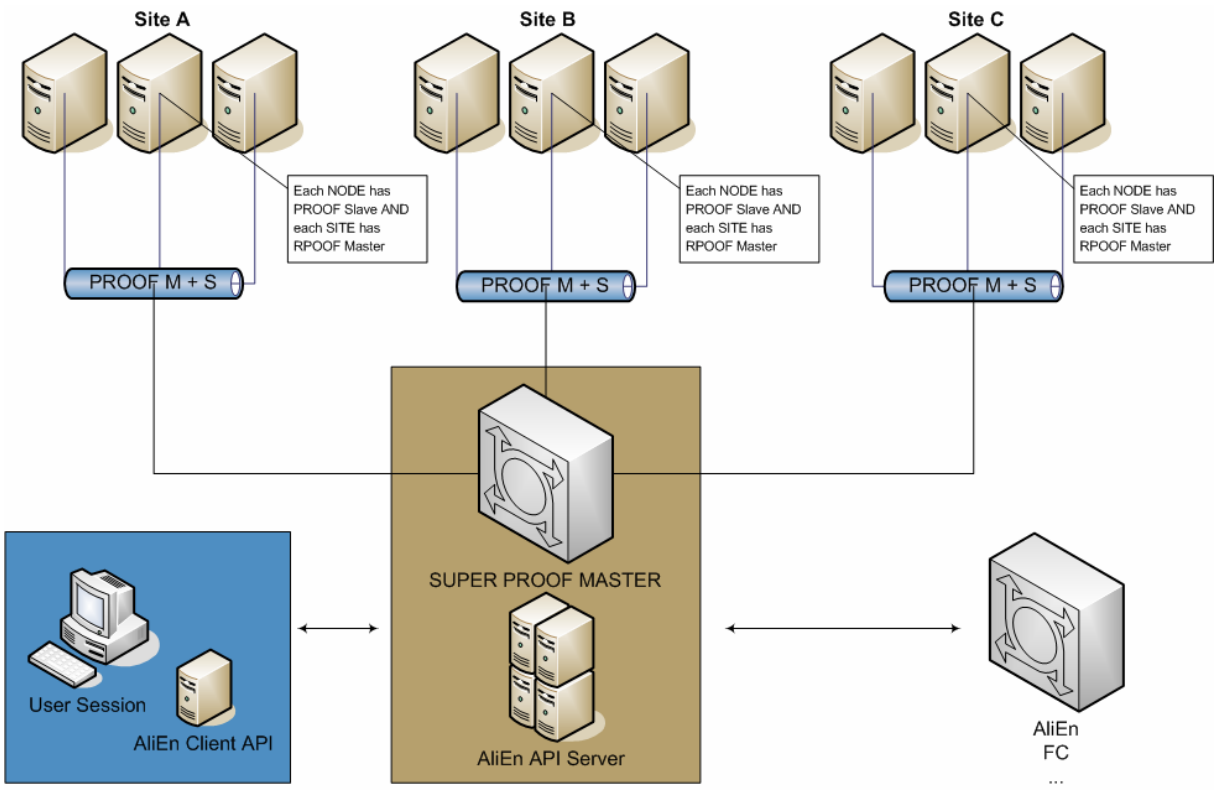


Figure 5: AliEn and PROOF

As a short description for the picture we should mention that every site provides at least one PROOF Master, each Worker Node of a site provides a PROOF Slave, one of the powerful Grid's site (probably a Tier 1 site) should provide access to the SUPER PROOF Master, which will manage and distribute jobs to the client sites. The actual retrieving of data files, submission of jobs, as well as security authentications are managed by the AliEn Grid and are

used by the PROOF system and end-users. A connection to the PROOF daemons is provided with the help of a TCPRouting Service at every site.
Users will just submit a normal job packet to AliEn; then AliEn and PROOF will do all the necessary work of distributing the job, processing and merging the result, and retrieving it back to the user or register it in the AliEn File Catalog by request.
Taking into account the advantages of the ALICE distributed data analysis we decided to accept this model as a strategic one for our gap analysis and development.

3.1.3 ALICE VOBox and gLite

We would like to enable or let's say spread the ALICE like analysis for a wide range of Grid middleware, or at least to proof that it works in general and for the most popular middleware flavors, other than AliEn.
Currently in order to process AliEn jobs and to have an interface between AliEn and the standard Grid middleware of all 4 LHC experiments, gLite, ALICE is using the so-called VOBox, as it is provided by the LCG project and EGEE. For example, to process AliEn jobs on a gLite site, administrators must install and use at least one VOBox node with a Grid to Grid interface of the AliEn software installed on it.
Unfortunately we found that it is not so far possible to provide ALICE-like distributed analysis as it is and without an AliEn client present or an ALICE VOBox installed. This is the one major disadvantage, which prevents distributing the modules and to bring it to the different experiments.

However, our task is to analyze and to implement a way, which could provide a possibility for different physics communities and not only physicists from ALICE to use Grid middleware for all the tasks needed.

4 Gap Analysis

4.1 Model description

In order to achieve our aim we need to create and investigate a system, which allows users to process a parallel and distributed analysis. The final system must be flexible and documented enough to be ported to different Grid middleware flavors with minimum requirements on human and time resources. We have to provide an operational test model which would be based on one of the popular and the most used Grid middleware flavors as well as to provide a precise documentation with a model description (including source code and its description) and recommendations for further developments, like investigations for bottlenecks and their descriptions and hints.

By achieving our goal we are going to implement and test a model, which will give a possibility of implementing an efficient distributed parallel data analysis based on any Grid middleware (as an exemplary case we choose the agreed standard Grid middleware of the 4 LHC experiments, gLite) and which could proof the possibility of creating a model for a generic analysis on the Grid for the HEP community.

4.1.1 The Model's "road map"

All of the following steps are closely described in the paragraph "§5 Design concept and development plan". Here we will just make a short list of required steps of accomplishments for the model.
The modeling consists of the following steps, which must be accomplished:

1. The following is, so to say, the input data for our model. A proper development environment must be provided. This environment must include access to the desired Grid or otherwise a local Grid test-bed must be installed. Also access to the Grid User Interface must be provided as well as access to the Grid middleware API. So far our model is based on the C++ language; therefore the Grid middleware should provide a C/C++ API.
2. An interface to the existing middleware has to be developed and its use in the experiment frameworks has to be enabled. As it has been already stated, ROOT is a pretty much standard framework for all of the HEP experiments. To accomplish the task, in particular, one should design and implement a ROOT plug-in of the desired Grid middleware. By accomplishing this stage one gets access to all the required functionality of Grid and ROOT. This is a big and an important step for the model, since a huge part of the distributed analysis, and analysis at all, is implemented by ROOT itself and a proper design and implementation of this plug-in will affect the quality of the whole analysis process and its results.
3. A local PROOF cluster has to be installed and provided.
4. Possibilities for executing PROOF master and PROOF slave processes on the gLite Grid have to be investigated.
5. A concept of using PROOF master und slave processes on the gLite Grid has to be implemented.
6. PROOF job submission on the desired Grid middleware (gLite as a test environment) has to be implemented.

7. The whole system has to be tested, i.e. a test analysis code on the desired Grid middleware using the constructed model in full operational mode has to be created.
8. The system has to be deployed to test users. This also consists of a good documentation, support of the software and an intensive dialog with users in order to advance this product.
9. The system has to be deployed for public use. After preproduction – test deployment, comes a time for public release. At this stage we are going to apply our model and software to become officially accepted and included in the ROOT framework.
10. A ROOT plug-in for several different Grids has to be implemented, to proof the usability of the interface.

4.1.2 The Model's criteria for success

The model can be considered as a successful one, when the following points are accomplished:

1. At least the first 6 steps from the model's "road map" (see §4.1.1) have been accomplished successfully.
2. The project has a clear and detailed documentation. The documentation must be divided in two parts, the first for end-users, and the second part for advanced users – application developers. The source code of the project must be documented as well.
3. One type of a TGrid like ROOT plug-in next to the AliEn implementation became part of the standard ROOT installation.
4. On of the major criteria for success is that the concept should be used by several different physics collaborations. Popularization and simplification of the Grid enabled Interactive Analysis is a main goal.

4.1.3 Development Environment

4.1.3.1 Grid middleware

We decided to base our project on the gLite Grid Middleware, since it is widely used and one of the biggest operational Grid middleware flavors today. gLite is a strategic Grid middleware for HEP.

gLite (pronounced "gee-lite") is the next generation middleware for grid computing. It was born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as a part of the **E**nabling **G**rids for **E**-scienc**E** project (EGEE). The EGEE is Europe's flagship Research Grid project and the world's largest Grid infrastructure of its kind. The Project gLite provides a bleeding-edge, best-of-breed framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet [3].
At the time of writing, the EGEE project was involving more than 70 partners from 27 countries, arranged in twelve regional federations, and providing more than 20,000 CPUs, almost 200 sites and 10 Petabytes of available disk storage, and access to Mass Storage Systems at a number of sites. This infrastructure supports 7 scientific domains and more than 20 individual applications.
The architecture of gLite is shown in the Figure 6.

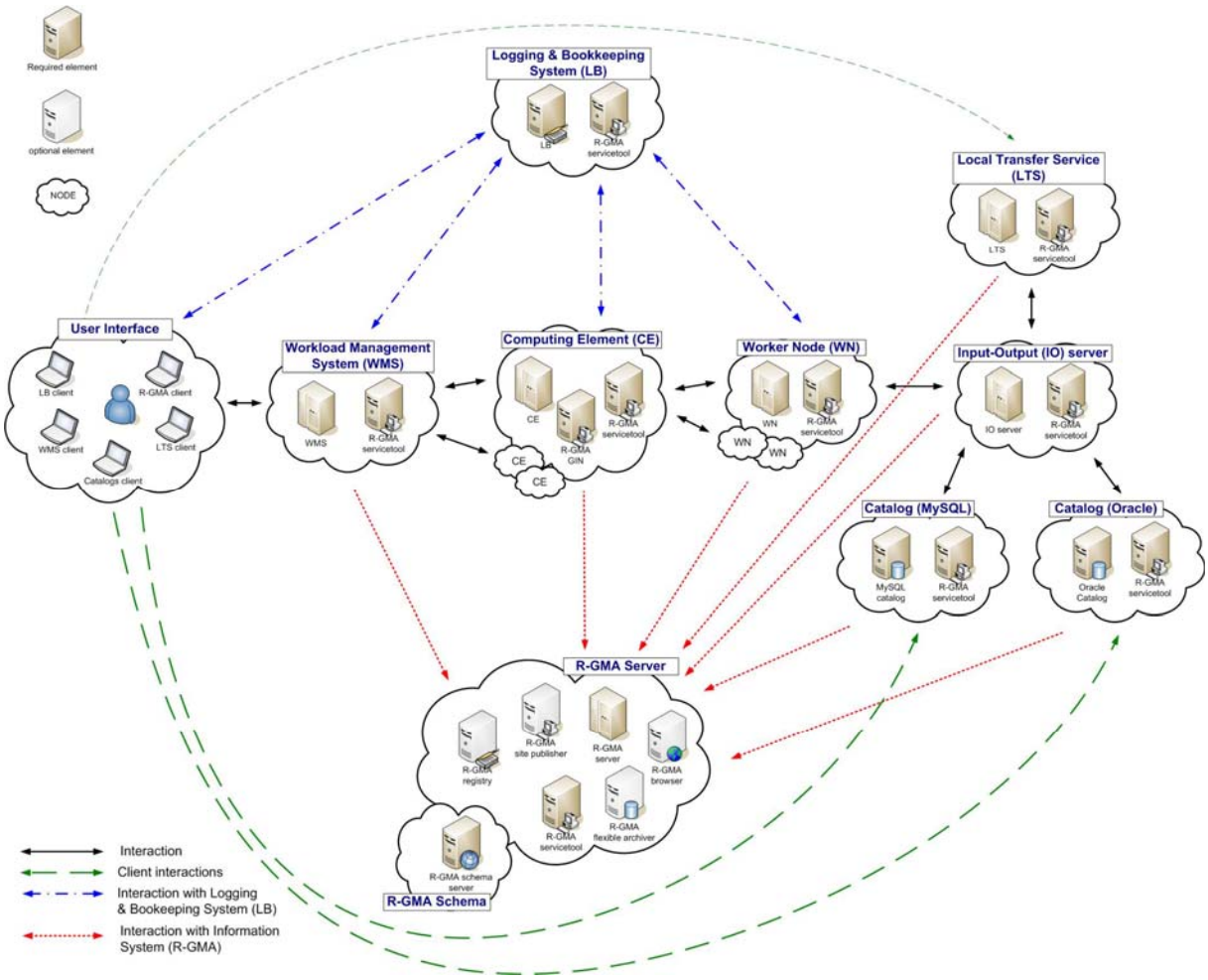


Figure 6: gLite Service Deployment Scenario [3]

The development environment of our gap analysis is based on our experience with gLite R1.4, R1.5 and R3.0. Those versions of the middleware were at GSI locally installed, configured and used. We investigated carefully most of the components of the middleware and its internal structure. This is a basic and an important analysis which should be done before coming to the development of an application which will use the API of the middleware.
In our development we tried to use the maximum number of application programming interfaces (API) which are currently provided by the most recent version of gLite in order to make a comparison of the components in a spectrum of usability, documentation, efficiency etc.
For example, a current implementation of our project is based on:

- gLite WMSUI – for job submission, status querying and output retrieving;
- LFC (Local LCG File Catalog) API – for file catalog operations;
- GFAL (Grid File Access Library) API – for file operation on the gLite Grid.

But we also used the following components of gLite:

- WMProxy – as an alternative for job submission -;
- FiReMan – for file catalog operations;
- gLite I/O server — for file operations on the gLite Grid.

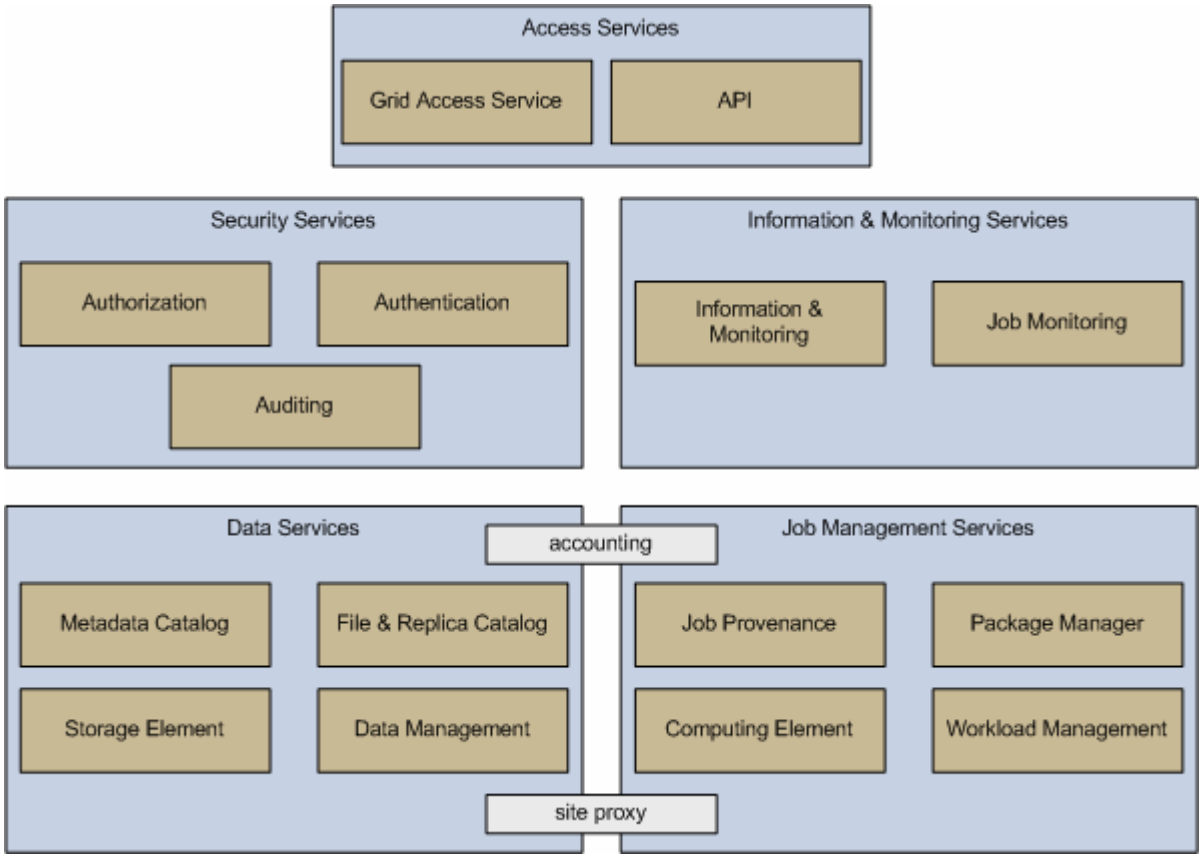


Figure 7: gLite Services

4.1.3.2 ROOT

The main part of our project is a ROOT plug-in for gLite. This plug-in must be designed and developed in terms of our model.
During the development process we used several versions of ROOT ranging from v5.10.00 to v5.12.00f.

4.1.4 Design and implementation of the ROOT plug-in for gLite

Since we decided to try and implement an ALICE like model we were required to design and implement a very important part of the model, namely a ROOT plug-in for the chosen Grid middleware.
At this stage of the project we have designed and developed a ROOT plug-in for the gLite Grid middleware. This is an essential part of the model!

As a first step we have analyzed and investigated the gLite API. We wrote a lot of small tests in order to get “a feeling” and to understand restrictions of the API. That was our pre-development investigation.
Then we had to investigate the Grid Interface provided by ROOT, since our plug-in was supposed to be an implementation of this interface (see Figure 8).

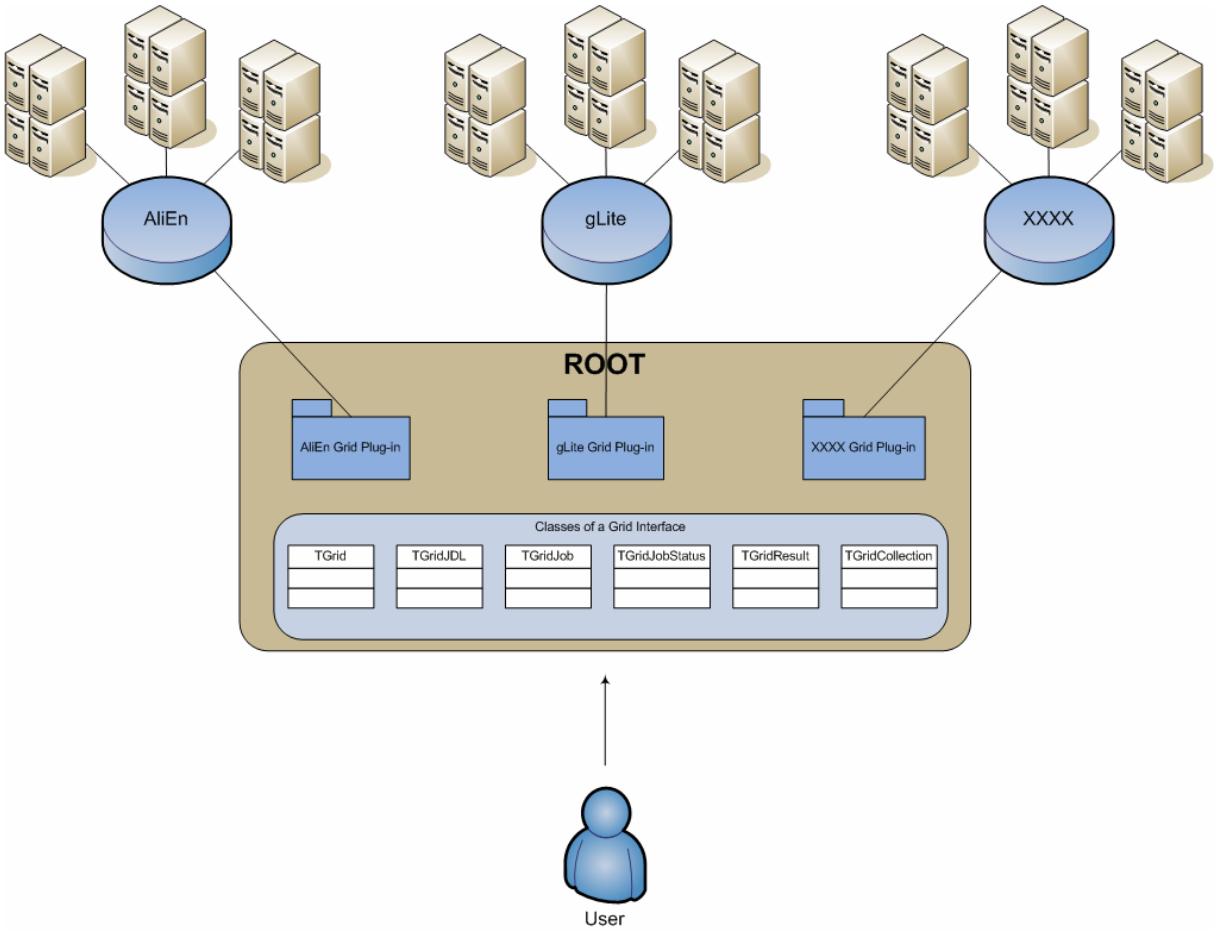


Figure 8: ROOT Grid Interface

Our implementation of the plug-in consists of two parts. The first one is the design and implementation of the so-called gLite API Wrapper (see Figure 9). The second one is the implementation of the plug-in itself.

4.1.4.1 gLite API Wrapper

A gLite API Wrapper (GAW) is a library, which wraps several gLite API modules, implements automation, and represents the interface for the user. Through the usage of GAW the user gets a general interface to the gLite middleware.
Our ROOT plug-in is using the interface of GAW to access the needed functionality of the gLite middleware (see Figure 9).

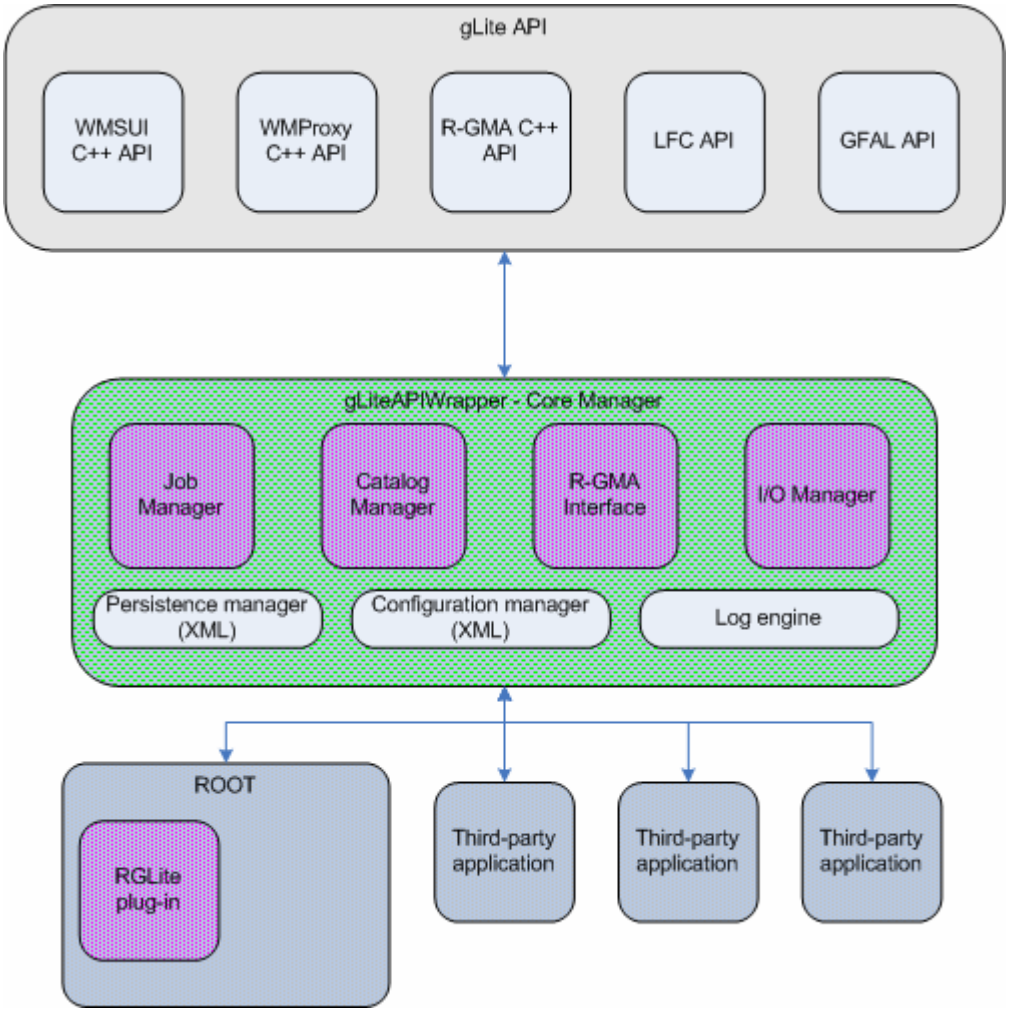


Figure 9: gLite API Wrapper and RGLite plug-in

One of the main reasons why we started to design this library is to keep the modularity of our system. The gLite API is very complex, and one of the disadvantages of it is that it is very not consistent and doesn't have a common "design line". That means gLite API consists of several modules (like several WMS APIs, LB API, LFC API, etc.). Interfaces of these API modules are not standardized and have no general schema, which makes these API modules difficult to use. Also, the important thing is that the gLite API modules require many secondary libraries they depend on. And when one wants to accumulate in one application the functionality of different API modules, a build configuration (a Makefile) becomes very complex, and the overall structure of the application becomes very difficult to understand. Dividing our ROOT plug-in into two general parts helps us in the development process, to easily write tests for both parts (while we have a strong border between the ROOT part and the gLite API part), and simplifies the deployment and support of the released software.

Thus we have implemented GAW as a library, which is released in a number of managers. The managers work directly with the gLite API modules and implement some API extensions (those extensions we can call as utilities or helpers or automations). This is done in order to extend the API functionality.

To simplify the development and the support of the project and to follow Open Source de-facto standard, we are using **GNU auto-configuration** tools for building and deploying GAW.

We consider that project documentation is the most important part of the project, in order to track code documentation we are using the **doxygen** engine [8]. Code documentation also facilitates latter support. The project documentation is kept and maintained in a central GSI WiKi, where the GSI GridTeam, and particularly the D-Grid project have a site [9].

The source code, tests, configurations, and some of the documents are tracked with the help of the Central GSI Sub Version System (**SVN**), where the GridTeam has several repositories.

Most of the GAW managers are **Singletons**, the implementations of which are releasing **Singleton Design patterns** and are done in C++ language.
We chose this schema to keep tracking of the objects and their status, and to avoid memory and code pollution. A Singleton's intent is to ensure that a class only has one instance, and it provides a global point of accessing it.

At the time of writing of this report the GAW library was consisting of the following:

- GAW Core Manager,
- GAW Job Manager,
- GAW Catalog Manager,
- GAW Persistence Manager,
- GAW Configuration Manager,
- GAW Log Engine.

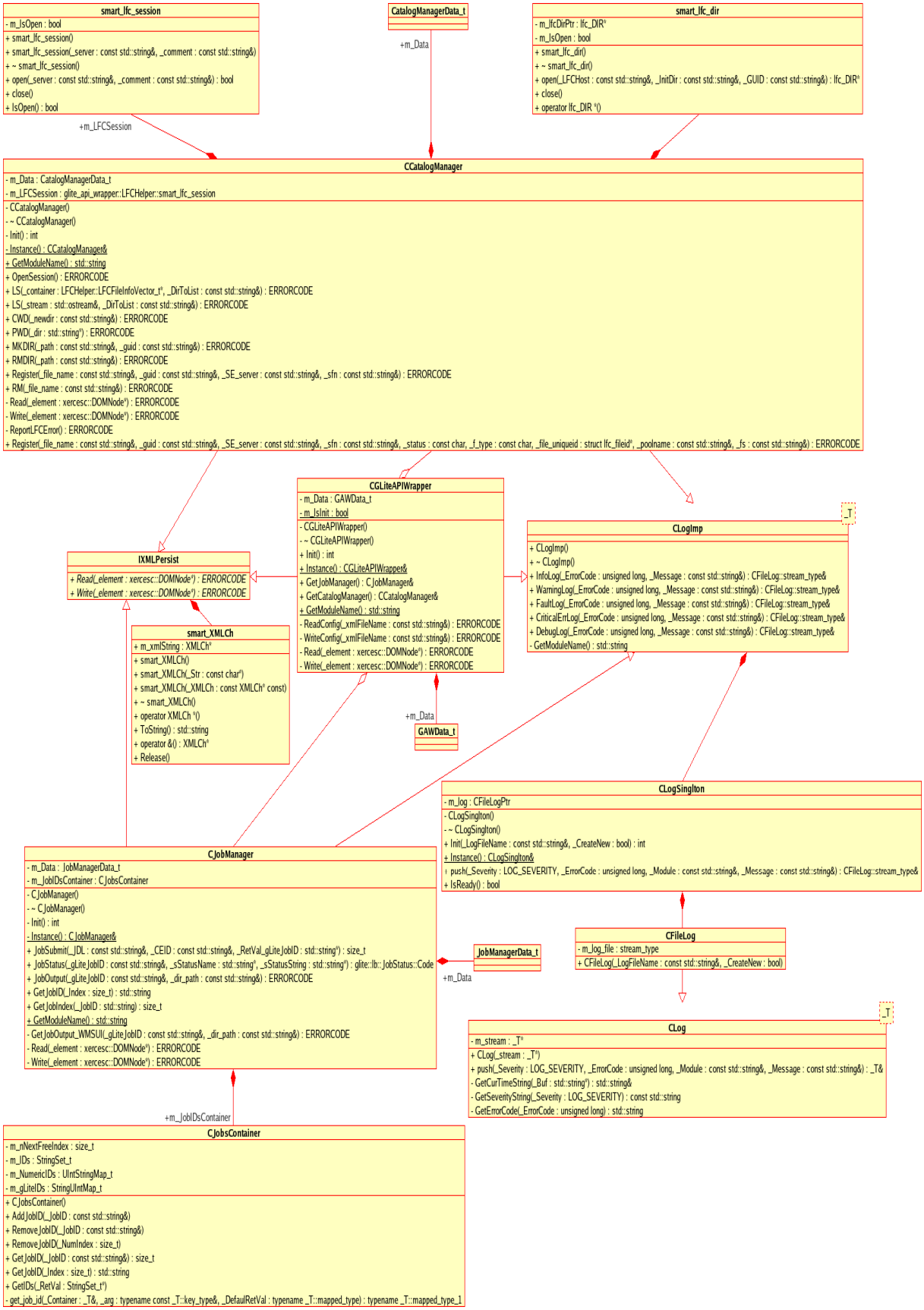


Figure 10: UML class diagram of the GAW library

4.1.4.1.1 GAW Core Manager

The Core Manager is an interface of the GAW library. This manager is responsible for configuration, the proper initialization of sub-managers (like Job manager, Catalog manager, etc.), and also partially responsible for a persistence of all "top-level" managers of the library. Normally The Core Manager is the first which should be initialized by the end-user.

4.1.4.1.2 GAW Job Manager

Using this manager a user is able to process job operations on the gLite Grid such as job submission, query of job status, and output retrieval.
The manager also keeps track of all processed jobs. All jobs have got at least two types of IDs, one is a gLite JobID, which comes from the EU DataGrid Project and is of string type, and the other is a numeric Job ID – job index, which is done for compatibility reason, see §4.2.1.2.
The Job Manager supports a job persistence mechanism. That means that after the user processed several jobs and wants to close the GAW session, the GAW Job Manager will store all required information about the current status in order to be able to restore the session next time. This feature can be switched on or off by the user with the help of the GAW configuration.
Currently the Job Manager is implemented as a Singleton object and it uses only the gLite WMSUI API. In the future we want to make this manager a bit more complex; we want it to support several different gLite API modules for a job submission. Namely, we want to implement not only WMSUI API job submission, but also WMProxy API [10], which is more modern and has many advantages in comparison with WMSUI. The implementation which uses WMProxy API is in our development plan for the next stage of the project. We found that the functionality of WMSUI API is enough to prepare and test a prototype of our model. And taking into account that WMSUI API is so far the most used way of doing job operations in gLite, we started our Job Manager from this API module. But for more serious work we would require functions which are available in WMProxy API, only..
The WMProxy client API supplies the client applications with a set of interfaces concerning the job submission, and control services made available by the gLite WMS through a web service based interface. The API provides the corresponding method for each operation published in the WSDL description of the WMProxy service. The request types, supported by the WMProxy service, are:

- Job: a simple application;
- DAG: a direct acyclic graph of dependent jobs;
- Collection: a set of independent jobs;

Jobs in turn can be **batch**, **interactive**, **MPI-based**, **checkpointable**, **partitionable** and parametric.
By our development plan the Job Manager may be a bit redesigned in order to support modern C++ technique, for instance template type traits. So it could be instantiated by different types, which will wrap different gLite Job controlling APIs, so the user gets a possibility to simply use both WMSUI and WMProxy APIs. Probably most of the GAW managers will be adapted for this mechanism in order to extend the functionality of the objects and to support and get the possibility to analyze and test bigger ranges of gLite API modules.
But this way is still under development and must be carefully investigated. We should compare the advantages and disadvantages based on user and project result requirements. It is

not so far obvious that we need for our model to support all of the possible gLite API modules.

4.1.4.1.3 GAW Catalog Manager

The development of the Catalog Manager appeared to be more complicated and time consuming than it was expected to be. The story was the following:
we used the "JRA1: Data Management Documentation page" [11] and "JRA1: Data Management site" – the official gLite Data Management contact, to design and to develop our Data Management Components. Therefore the development was based on two major components of the gLite I/O system, namely the gLite I/O Server and the gLite FiReMan File Catalog. They have been stated as the main gLite Data Management modules by JRA1. But unfortunately during our close work with the gLite I/O Server and the FiReMan Catalog it was discovered that those components are going to be excluded from future releases of gLite. That was definitely the fault of obsolete WEB information on the official gLite Web Site. If we would not have discovered several bugs in the Data Management modules and would not have posted them, we wouldn't have started the discussion with the gLite Deployment management which revealed that the official Web pages were not properly updated (see §4.2.2.1.6).
Now our Catalog Manager uses LFC. LFC is an official gLite File Catalog.
The GAW Catalog Manager implements the following operations:

- to set and query current the working catalog directory:
- to list the content of the selected catalog directory (list files, directories and their status);
- to create a file in a Catalog namespace;
- to add replica(s) to a given file in the Catalog namespace;
- to remove replica(s) from a given file in the Catalog namespace;
- to remove a file or directory from a given file in the Catalog namespace.

Combined with standard File Catalog interface functions the GAW Catalog manager implements a number of extensions (automation) to the normal catalog operations offered for a user, which are not present in the original LFC API:

- if a user requests to remove a directory and its content, the GAW Catalog Manager will remove all files of the directory and corresponding replicas,
- if a user requests to force removing a file, GAW will remove all its replicas automatically,
- we found the necessity to set LFC Host manually for the LFC API functions inconvenient (see §4.2.2.2.5). GAW tries a bit to simplify this by defining the key in its configuration file,
- most of the LFC API functions have a very short connection time out. The GAW manager implements a workaround (invisible for a user), in order to keep the connection during long user sessions.

4.1.4.1.4 GAW Persistence Manager

One of the Core GAW engines is the Persistence Manager.
Its main responsibility is to be a provider of persistence for the GAW managers. The GAW Persistence Manager uses an XML based file for storage. Some of the GAW managers use the

provider to store and restore the required information in order to be able to save and recover a user session's stat.
For example, the GAW Job Manager with the help of the Persistence Manager keeps track of a user's jobs.

4.1.4.1.5 GAW Configuration Manager

Another Core GAW engine is the Configuration Manager. It is not really a manager because its functionality is distributed between other GAW Managers. The Configuration Manager is only providing the interface and all of the GAW components which we want to have configured must implement this manager. The coordination of the configuration of GAW components is implemented in the GAW Core Manager.
The configuration is tracked in an XML formatted file. The currently supported XML schema of the configuration file is shown in Table 1. This is a preliminary version of the configuration file. We are going to extend it and to provide maximum configuration flexibility for the users with adequate default values in order to let the users configure the software as precisely as needed. Also we are going to provide a tool, which will automatically scan a gLite User Interface configuration and prepare a default GAW configuration file. The development of the tool is scheduled to the next stage of the project (see §5).

Table 1: XML Schema of the GAW configuration file

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="gaw_config">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="gaw_mng">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="config">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="logfile" type="xs:string" use="required" />
           <xs:attribute name="logfile_overwrite" type="xs:unsignedByte" use="required" />
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="job_mng">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="config">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute name="ns_host" type="xs:string" use="required" />
           <xs:attribute name="ns_port" type="xs:unsignedShort" use="required" />
           <xs:attribute name="lb_host" type="xs:string" use="required" />
           <xs:attribute name="lb_port" type="xs:unsignedShort" use="required" />
           <xs:attribute name="ce_id" type="xs:string" use="required" />
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
       <xs:element name="jobs">
        <xs:complexType>
```

```
            <xs:sequence>
              <xs:element name="job">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="glite_jobid" type="xs:string" use="required" />
                      <xs:attribute name="numeric_jobid" type="xs:unsignedByte" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="catalog_mng">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="config">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="lfc_host" type="xs:string" use="required" />
                <xs:attribute name="lfc_session_comment" type="xs:string" use="required" />
                <xs:attribute name="lfc_wrkdir" type="xs:string" use="required" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:attribute name="last_update" type="xs:string" use="required" />
  <xs:attribute name="version" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
</xs:schema>
```

4.1.4.1.6 GAW Log Engine

Several years of experience in Grid middleware deployment (installation and configuration) and the fact, that it is very complicated to trace an error in distrusted applications, make us to be concern on the message, that the logging is very important part of Grid middleware admin- and user-interface.
Our application provides the Log Engine, which used by the all of the components of GAW.
The application provides the following typed of log messages:

- **LOG_SEVERITY_INFO,** which indicates a normal information message;
- **LOG_SEVERITY_WARNING,** which indicates a warning information message, probably an erroneous situation but recoverable and not dangerous one;
- **LOG_SEVERITY_FAULT**, which shows that GAW detected and error which is recoverable but could be dangerous, and it is better to fix it before processing further;
- **LOG_SEVERITY_CRITICAL_ERROR,** this type of message shows that the GAW faced an unrecoverable stat for the current procedure. Before going further user may want to fix the situation and the GAW log messages can help him to accomplish that;

- **LOG_SEVERITY_DEBUG,** this is only debug information of GAW functions, useful for the developers and for advanced users.

Currently GAW writes the log to the plan formatted-text file. But an internal architecture of the log engine allows extending it to the other sources, like relational data bases, console or some other clients of log messages.

4.1.4.2 RGLite plug-in

RGLite – is a ROOT plug-in for gLite Grid middleware.
In the same way as it is done in AliEn middleware plug-in we have to realize the ROOT Grid Interface (see Figure 3) but for the gLite middleware. Current implementation of the ROOT plug-in for gLite presented in the Figure 11, it is an UML class diagram of the plug-in.

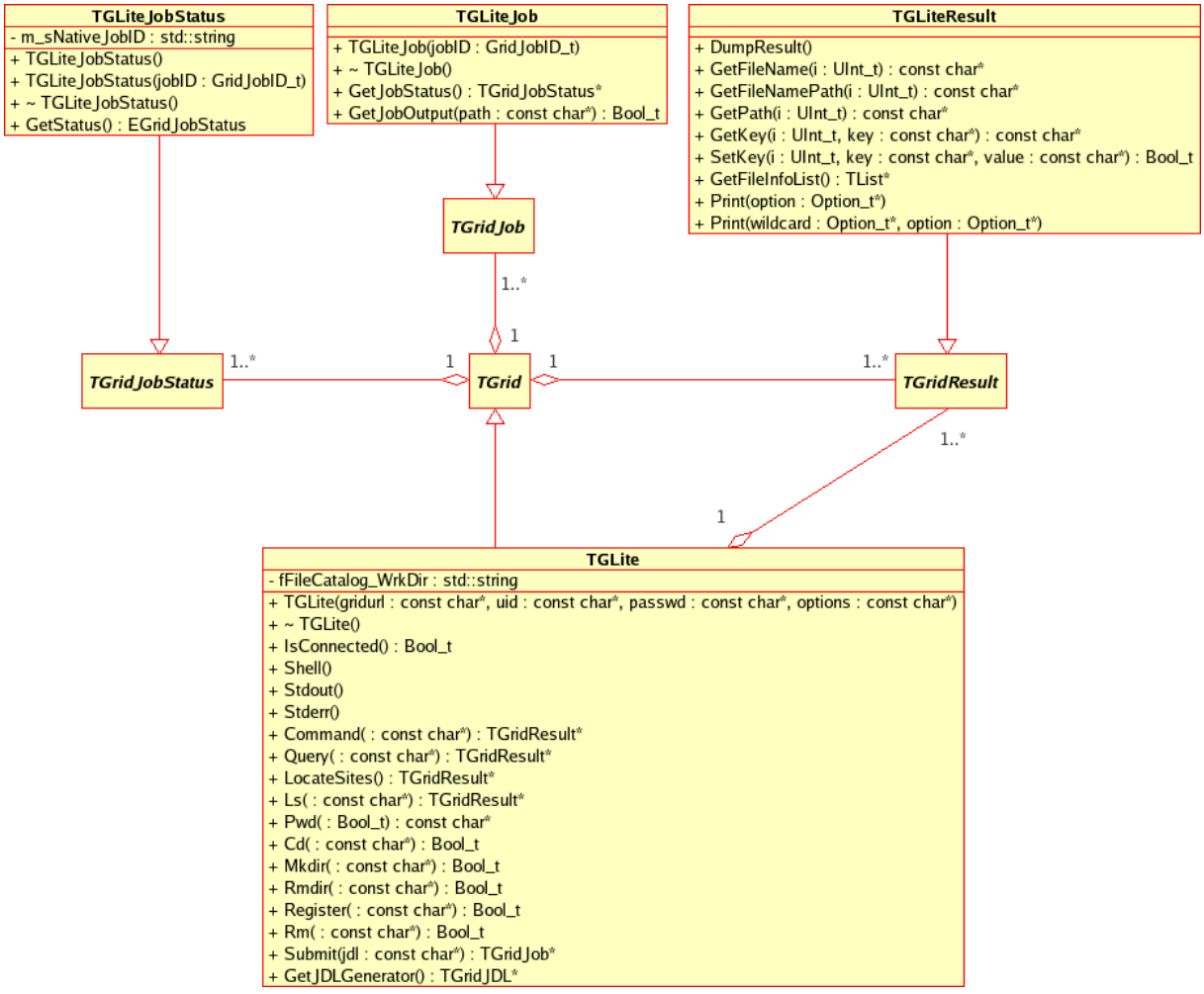


Figure 11: UML class diagram of the ROOT plug-in for gLite middleware

Current status of the plug-in is: completely finished development stage; this includes ready for alpha release ROOT plug-in for gLite, ROOT style plug-in documentation and a patch for resent version of ROOT. The patch makes an installation process of plug-in easer for a user.

Later we expecting our plug-in to be included to standard ROOT installation, but so far the patch will help to users.
The following functionality is implemented and available for the end-user by the current prototype version of the gLite plug-in:

- Workload Management System (WMS) operations:
  - job Submission,
  - job Status Queering,
  - job Output retrieving.
- File Catalog operations:
  - to set and query current working catalog directory,
  - to list content of the selected catalog directory (list files, directories and they stats),
  - to create a file in a Catalog namespace,
  - to add replica(s) to a given file in Catalog namespace,
  - to remove replica(s) from a given file in Catalog namespace,
  - to remove a file or directory from a given file in Catalog namespace.
- An Executive Logging.
- Support of an External Configuration.
- File I/O operation is could be performed with help of *TGFALFile* a ROOT class-wrapper of GFAL.

Since standard File operation mechanism in gLite is the GFAL and the GFAL class-wrapper is a part of standard ROOT installation, we don't need to implement this in our plug-in. Users can use standard ROOT implementation of *TGFALFile* (see an UML class diagram on Figure 12).

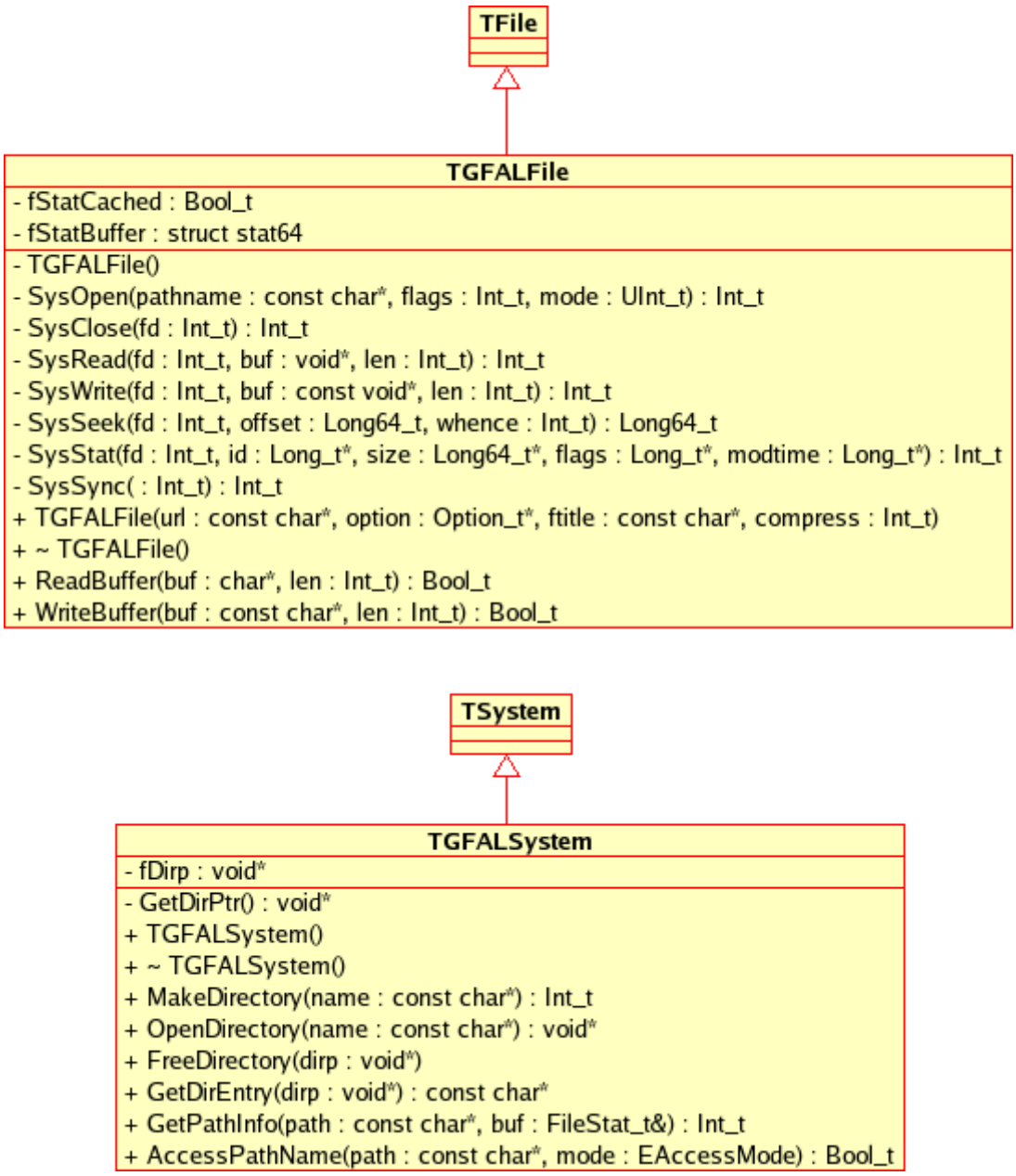


Figure 12: UML class diagram of the ROOT implementation of GFAL

4.1.4.2.1 RGLite Tests & Samples

In this paragraph we would like to present several “real-life” examples — test cases, which will show basic usage and abilities of the RGLite plug-in in very simple manner.
The first example (see Table 2) shows us some of the basics operations provided by ROOT Grid Interface and RGLite plug-in for gLite middleware. The first example consists of the following steps:

1. Connection to the Grid MW. In this step user needs to initialize Grid plug-in; one can make so by calling method “Connect” of the *TGrid* class with according parameters. In our case we call connect with string “*glite*” as a Grid URL string,

which means that we want to access gLite Grid. According to the given parameters ROOT will initialize proper plug-in if available.

2. Submitting a job using abstract Interface *TGridJob* and method Submit.
3. User can check a status of the jobs by using abstract Interface class TGridJobStatus and its method GetStatus.
4. Finally, when job is finished and has status "kDONE", user can retrieve the jobs output back to the User Interface. In order to do so user should use our extension to the abstract ROOT Grid Interface, namely, user should use *TGLiteJob* class, which is implementation of *TGridJob* with some useful extensions (it is registered as a low level gap in ROOT Grid Interface, see §4.2.1.3).

On the Figure 13 you can find the screen shot of a life ROOT session with processed code of example from Table 2.

Table 2: RGLite Usage Code example – Job submission, Status querying, Output retrieving

```
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Submitting a Job to gLite Grid
TGridJob *job = gGrid->Submit("JDLs/proofd.jdl");
// querying a Status of the Job
TGridJobStatus *status = job->GetJobStatus();
status->GetStatus();
// Getting a Job's output back to the user
TGLiteJob* job_glite(job);
job_glite->GetJobOutput("/home/anar/");
```

It needs to be mentioned that from end-user's point of view a usage of RGLite and AliEn plug-ins is mostly the same, since most of the time user will talk in "the language" of ROOT Grid Interface classes and in some rare case will use the specific extension of the plug-ins.

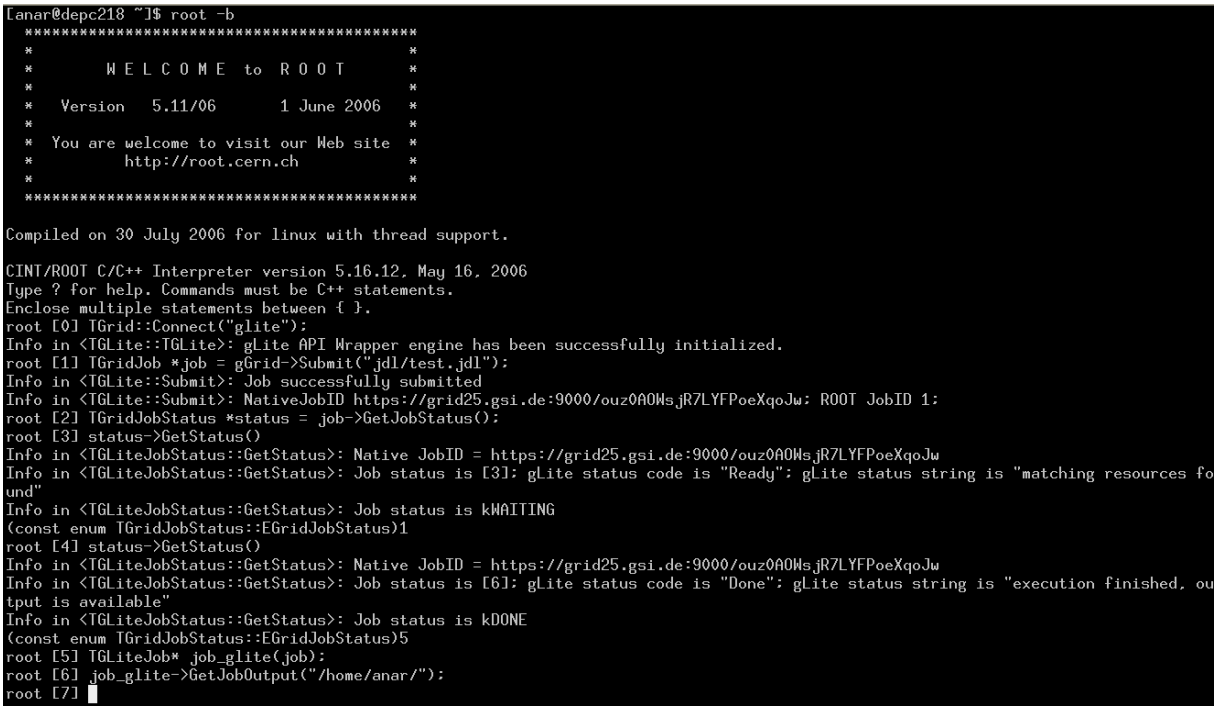


Figure 13: Life example of using RGLite (executing of the code from Table 2)

The examples on Table 3, Table 4, Table 5 and Table 6 show how an end-user can handle File Catalog operations using *TGrid* interface and RGLite plug-in, such as querying and changing current catalog directory, listing files and they full or partial information, creating new catalog directory, etc. The examples below are designed for ROOT C++ interpreter but can be also used in normal C++ application with enabled ROOT support.

Table 3: RGLite Usage Code example – Changing File Catalog directory, querying lists of files

```
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Changing current File Catalog directory to "dteam"
gGrid->Cd("dteam");
// Querying a list of files of the current FC directory
TGridResult* result = gGrid->Ls();
// Printing the list out
Int_t i=0;
while (result->GetFileName(i))\
> printf("File %s\n",result->GetFileName(i++));
```

Table 4: RGLite Usage Code example – List full file information of a File Catalog folder

```
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Changing current File Catalog directory to "dteam"
gGrid->Cd("dteam");
// Querying a list of files of the current FC directory
TGridResult* result = gGrid->Ls();
// Printing the list out, including full file information
result->Print("all");
```

Table 5: RGLite Usage Code example – retrieving a name of a current File Catalog folder

```
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Changing current File Catalog directory to "dteam"
gGrid->Cd("dteam");
// Printing a current working directory
std::cout << "Working Directory is" << gGrid->Pwd() << std::endl;
```

Table 6: RGLite Usage Code example – creating a new File Catalog Folder

```
 // Initializing RGLite plug-in
TGrid::Connect("glite");
// Changing current File Catalog directory to "/grid/dech"
gGrid->Cd("/grid/dech");
// Creating a new File Catalog Folder
Bool_t b = gGrid->Mkdir("root_test2");
```

4.2 Low-level Gaps

We have investigated gaps in all components; those were used in the design of the Model. The gaps which are listed below are low-level gaps and divided on several parts.

4.2.1 ROOT Framework

4.2.1.1 Grid Interface

The ROOT provides an Interface for a Grid. This is a kind of a gap the fact that a design of the interface is mainly based on knowledge of AliEn Grid middleware. One of our goals is to improve this interface, so to make it suitable for wider range of the Grid middleware.
All the following gaps are going to be discussed with ROOT and AliEn developers.

4.2.1.2 TGridJob::JobID

A Class member *TGridJob::JobID*, which has a numeric data type, needs to be revised. This class was designed on AliEn example, where *JobID* is numeric. But while developing gLite plug-in for the ROOT we found, that this is very inconvenient having *JobID* predefined in the parent class and as numeric. For example gLite middleware uses DataGrid Job ID, which is unique string ID. So, either TGridJob should be a template class interface, which leads to a slight redesign of whole *TGridXXXX* interface or type of *TGridJob::JobID* should be changed.

4.2.1.3 GetJobOutput is missing

There is a missing functionality in the *TGridXXXX* interface.
We didn't find a way (method) which could retrieve job output directly to UI (or machine, where ROOT instance is running, local folder, for instance). There are only possibilities to register job output in a File Catalog and get it to there (AliEn way).

4.2.2 Third-party Gaps

As a third-party gap we would like to call the one that was discovered and analyzed by our effort, but unfortunately can't be fixed or redesigned by ourselves. Mostly it is a problem, a bug or a possible improvement which we traced and discovered and which related to gLite middleware itself or to one of its components. We always tried to be in contact with gLite development and deployment team since we choose this middleware for our project.
In this terms "contact", means, that as soon as we discovered, traced a bug we posted it in issues tracker or requested an improvement of a component of the middleware etc. We were very intensively working with the middleware.
For the period of the first stage of the project we have found and reported to CERN Savannah issue tracker [7] more than 15 confirmed bugs in released (production) software, and some of the bugs were blockers in general or specific cases. We not only posted the bugs, but also all of the bug tickets we tried to provide sufficient information, based on our investigation, including workarounds and possible fixes.

Our site is actively participating in testing of the middleware. We also helped to several different sites to install and used middleware and helped to resolve some issues, since, for example, GSI was on of the first site who installed R1.5 release, which was having several bugs (blockers).
We are Grid community, and it is our role to help to populate the Grid technology and make it user-friendly and open for an end-user.
In despite of the fact that the following gaps can't be really fixed by us and only can be reported to the gLite development and deployment teams, we would to list them here in this paper, because a huge amount of work have been done on middleware and on API investigations and analysis.

4.2.2.1 gLite middleware general

4.2.2.1.1 A gLite test-bed installation and configuration

In order to prepare environment for our project gLite R1.4 (preproduction release), gLite R1.5 and then gLite R3.0 were installed and configured, what gave us test and development environment. Unfortunately it must be mentioned that approximately 60% of our project time were spent on middleware. For the first R1.4 and R1.5 had almost no support, because most of the power of gLite deployment and development team spend on future integration release (LCG MW + gLite = gLite R3.0). The other think is that middleware was containing bugs, which were blockers and until we resolve those bugs and find a workarounds or fixes we couldn't continue our project. There was and still is a lack of low-level architecture documentation of middleware. There is some lack of documentation which describes extended instillation and configuration of middleware – a different way as default one. Plus some more problems described below.
It is also worth once more to mention that GSI was the one of the few sites (and the first site in DECH Region), which managed to install R1.5, while providing fixes for the bugs and helped to other sites and GSI's gLite R1.5 test-bed was the first D-Grid development environment. The R1.5 release had several problems which were revealing themselves in some cases during configuration and operation procedures.
Probably we can conclude that installation and configuration of middleware is not so far as easy as one would expect it to be. Grid is a very new technology and a role of everyone in this business is to develop it further and bring it to standardization.

While installing and configuring the middleware we accumulated valuable experience about internal architecture of gLite middleware, what is very helpful in our development and analysis of API, while there is still lack of low-level architecture documentation.

4.2.2.1.2 gLite and Operating Systems

Currently it is very difficult and officially unsupported to use gLite under different operating systems, but SLC3. Taking into account that gLite is build of components, which are portable, like CONDOR and GLOBUS, one can conclude that gLite must not be strictly dependent on SLC3 (SLC3 was the only supported platform and the time of writing of this paper). One should consider this is a priority issue, in order to populate Grid.

As far as we're concerned, gLite community is working on middleware porting issues…

For example, GSI actively participate in porting gLite to other Linux flavors, like we have ported UI (User Interface) and WN (Worker Node) to Debian ("Woody" and "Sarge"), Fedora Core 5 and SE (Storage Element) has been ported to MacOS X 10.

4.2.2.1.3 gLite internal modules and components

There is an obvious lack of documentation on the internal gLite architecture. We consider that it is very important to know for developers or administrator the internal architecture, the components on which gLite is based (like Condor, Globus, VOMS, etc.) and how they depend on each other. Without this knowledge it became a difficult task to trace possible or real problems on the site or develop software for this specific Grid.

4.2.2.1.4 gLite Logs

It is very important to have general information, which describes log books of gLite components. This must be a documented which is an extension of information for "gLite internal modules and components". Currently there are only some of gLite components' log books, which are documented.
Missing information about location and processes of logging and of defining log levels must be considered as important one!

4.2.2.1.5 gLite UI

A gLite deployment team should consider providing relocatable middleware UI distributive on primary bases. Currently it seems to be that the gLite UI relocatable tar-ball is supported on the lowest priority. But this is very important to give users possibilities to install Grid UI to any machine or OS, and make this process as easer as it could be possible with minimum configuration efforts required from the user. This relocatable tar-ball distribution must be up-to-date middleware package as all other important Grid packages.
Simply to say user must be able to have one click UI installation on his laptop which runs under Operating system XYZ and get access to desired Grid.

4.2.2.1.6 gLite Documentations and WEB sites

In such a world-wide distributed project as any Grid project, it is primary to have a WEB site – interfaces of the project updated on regular bases.
There was a very unfortunate incident, which affected our project. At the time when we started to develop I/O mechanisms of our GAW library and RGLite plug-in we used "JRA1: Data Management Documentation page" [11] and "JRA1: Data Management site" – official gLite Data Management contact, to design and develop on Data Management Component. There was time and man power spent on test, design and development of I/O part of GAW as well as on installation and configuration of gLite I/O server and FiReMan file catalog and configure them against our dCache storage. There were also several bugs discovered while working on this issue. Fortunately our tickets, which we opened to founded bugs, initiated discussion with gLite developers and which had revealed the fact that in the future releases of gLite the components gLite I/O server and FiReMan file catalog assigned as a deprecated and will be soon removed from the standard middleware installation and support. This "small" inconsistency of the real fact and fact listed on the official WEB site or documentation leads could lead to an expensive problem or mess and disorganization.
The conclusion would be the following:

The information WEB sites and documentation in such a distributed project **must** regularly be updated, at least when major release of software comes!

4.2.2.2 gLite API

4.2.2.2.1 API Installation

It will be better if instillation of gLite API became easer, or let us say convenient as it is now.
For example one wants to install WMProxy C++ API or any other API module. We concerned, that the package should provide its dependences list which isn't unfortunately always a case for gLite API modules and installation should be one step installation, like for instance just executing of "*apt-get install WMProxy_cpp_api*" and that's all. All the rest should be done automatically. Again, API module is better to be concentrated in one package which makes API installation packages a set of independent modules and with clearly defined external dependences list.
As an example, we would like to give a list of what generally a user should do in order to get WMS UI API installed on his machine now:

- One must be a root!
- One needs to install more than 15 different gLite API packages!
- One needs to install external dependences packages by himself.

This looks like a bit too much for just to be able to develop software for Grid.
Why don't just provide one package or at least 2: "XXXX_api_XXXX_**FULL**" and "XXXX_api_XXXX_**Compact**" or something like that, instead of putting a user in such a complicated situation, we are talking only about native gLite API modules here.
Anyway to compile just a simple job submission, one wouldn't need the entire API with all its modules so he can use "compact" package. On the other hand if one wants to develop an application using gLite API he can have it completely installed.
If it is not possible, for some reason, to distribute API modules this way, than installation helper (script or application) should be taking this job. Accessibility and usability of API of the Grid is one the way to populate Grid and make it used by people.

4.2.2.2.2 Library dependences

We found that API inter-library dependences are not documented. This documentation will be very helpful for one who wants to use API and link against it. Even so, some of API libraries are built with unresolved symbols inside, which is not always a good, for example in case of C/C++ API, which makes them rather difficult to be resolved without a clear documentation, which is currently missing.
We also found the dependences of gLite API on third-party libraries or modules are badly documented.
Besides taking in account complex installation of API modules (see §4.2.2.2.1) it became difficult to resolve all dependences, sometimes.

4.2.2.2.3 API modules dependences

There is no clear explanation (documentation) on API modules dependences. For instance, it is not clear which library (module) must be added to the project, if one wants to use WMSUI API functionality or R-GMA API. There some examples, which can be found on the Web, but

they aren't mostly up-to date and in addition they are distributed instead of being accumulated on one site (like gLite home or something like on gLite Developer Network) at least as links.

4.2.2.2.4 LCG API modules vs. gLite API modules

The gLite middleware consists of both types of API modules; some comes from LCG (**L**CH **C**omputing **G**rid) and some from gLite. Here we wont go to into much details, like saying that former DataGrid project or AliEn had give some significant amount of modules or codes to the gLite, but rather we will just point on general and obvious difference between LCG API modules and gLite ones.
So, in the current gLite middleware the both types (LCG & gLite) co-exist, some time ago they were merged.
For example, WMProxy API is a gLite API module which provides an interface to the job operations (this module was developed by gLite EGEE project), but LFC API is gLite API, which provides interface to the File Catalog operations and which was inherited by gLite from LCG.
While working with both types of API modules, it was noticed that there are a lot of differences.
Most of the gLite API modules normally are documented including doxygen style code documentation and have tests (small program which could be used to test the API) located in gLite CVS repository and which could be freely accessible by users. The documentation and code have coding and documenting style and standard, developed by EGEE project. It definitely gives an impression of industrial software development.
On the other hand some of the LCG API modules, like LFC API and GFAL, are not so good documented and it is pretty hard to find required information, sometimes it is so few, that it takes a lot of time to put API module on the needs – to use it. There is an obvious lack of examples and testes to LCG API modules. As an example we can give LFC API or GFAL API modules – two modules of high importance which belongs to LCG part. From the headers and some source code which we fortunately found, it looks like they have been "partially" inherited from CERN CASTOR project, but lack of own documentation and tests (or examples) and description of specific aspects of usage doesn't give a room for populating of them.

It must be clear stated, that API modules are going to be used by not only internal developers, but rather by end-users and/or by Grid application developers. The key points here are good documentations and availability of the source codes or at least availability of examples.

Since most of the gLite API modules are new ones. We can see the good attitude here. But still, old inherited API modules must be brought into a proper shape and documentation for a new one updated on regular bases.

4.2.2.2.5 LFC API, LFC server host name

Different functions of LFC API which are requiring LFC Host, taking LFC Host name in two major ways: some functions take LFC Host name as a parameter to them, or some functions require that an environment variable "LFC_HOST" should be defined. For example, to create an LFC session one should call "*lfc_startsess*" function, which is defined in the "*lfc_api.h*" as it follows "*int lfc_startsess (char *server, char *comment);*", where Name Server (LFC server name) should be given as the first parameter and in the same time some functions like "*struct*

*lfc_filereplica *lfc_listreplica (const char *path, const char *guid, int flags, lfc_list *listp)*" requiring server name as the environment variable.
It is very incontinent to have it in that way, especially to be forced to define any environment variable.
We would like to propose to allow developer to define Name Server host name as a parameter to the all function, which require a server name.

5 Design concept and development plan

In this paragraph we would like to shortly summarize what were done during the first stage of the project and mainly describe the design concept of the model, all the part of which are the result of gaps analysis and our research, which will be fulfilled in the next stage of the project. The model of a possible general Grid enabled distributed parallel data analysis for HEP community.

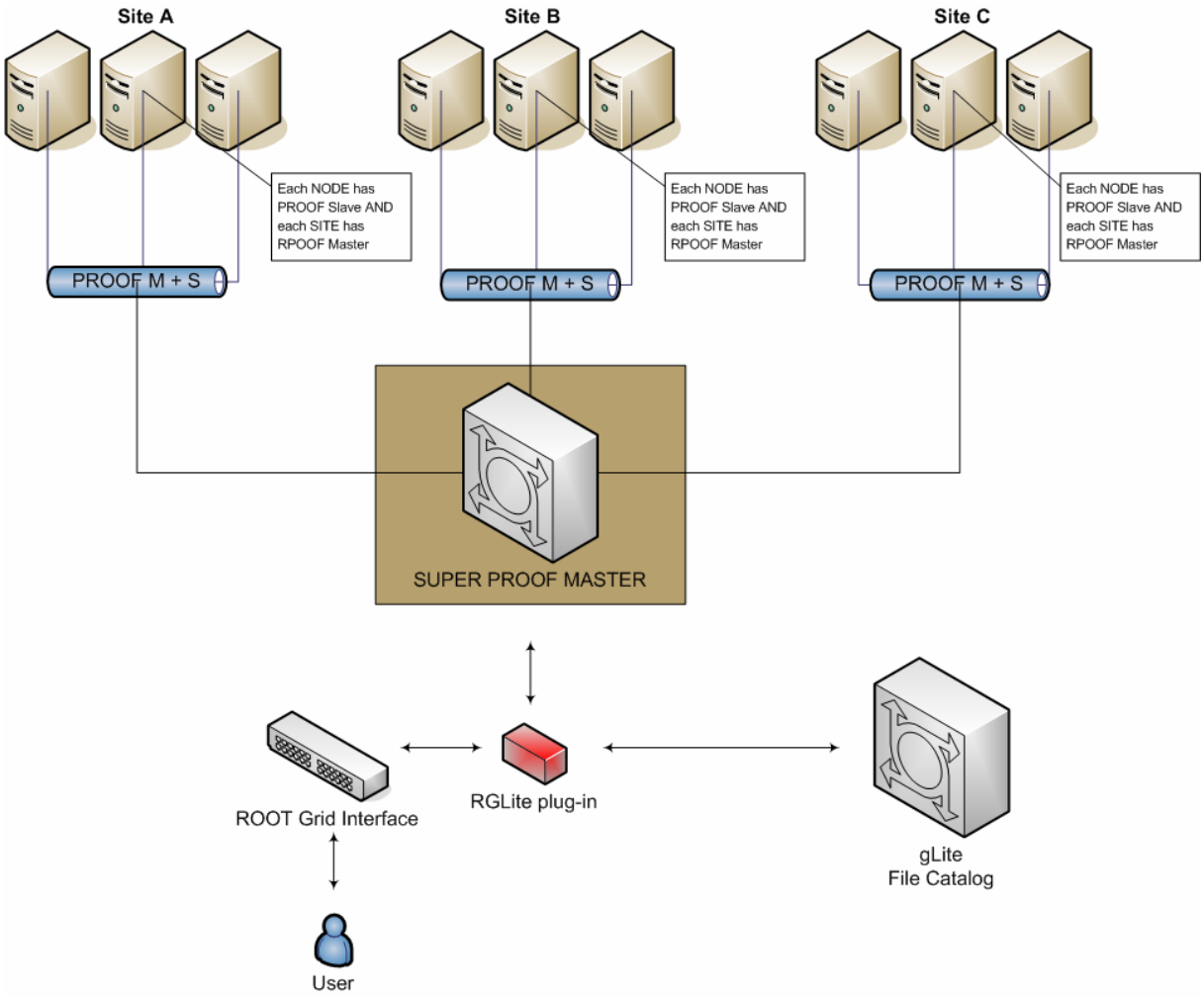


Figure 14: gLite and PROOF

High level picture of design concept of gLite Grid enabled distributed parallel data analysis is shown in Figure 14. The schema consists of the following parts:

- The gLite Grid Middleware — as a Grid provider, including gLite compatible I/O services (dCache, LFC, DPM, etc.). *Short status: This task is done and the testbed is available. Our team constantly working on this issue. We always have several gLite clusters: at least one for preproduction tests and one for the production. We also contacting with our colleagues in German/Swiss federation in order to exchange our experience and reuse clusters. We constantly keeping an eye on the*

current development and releases of gLite, since it could affect a plan of our development.

- Enabling PROOF in gLite environment.
  - PROOF services. Each worker node of each site should run PROOF slave and there should be at least one PROOF master for the site. *Short status: This is done only partially and the most of the work is going to be finished in the next stage of the project.*
  - There should be SUPER PROOF MASTER running somewhere. *Short status: This is not done. The work is scheduled to the next stage of the project.*
- ROOT Grid Interface with available gLite plug-in – this is essential component of the system, which enables user an access to a desired Grid. *Short Status: The task is done. In the next stage of the project this component is going to get some improvements and "beautifications", for example, in order to support several Job Managers types and some user-friendly changes of the interface (see §5.3).*

So far results look promising. But as it can be seen we are not still completely sure that there is a possible to create ALICE-like analysis based on different Grid middleware (namely the current research is based on gLite), because we have analyzed and implemented only a part of our model. What is done up to now:

1. A preliminary recognition, analysis of environment and conditions, which helped us to define the research and development strategy.
2. The definition of the model.
3. The gap analysis of chosen environment and of the model.
4. The implementation of components of the model (whole "ROOT" part is done).

Generally we can say that our plan for the next step of the project is to get design of the Model and is to get its implementation done in all aspects and to bring the system to a deployment.
Missing (not researched or not implemented) parts of the system shortly could be summarized as the following:

1. Local PROOF cluster.
2. "Marriage" of the PROOF and gLite.
3. Complete GAW and RGlite plug-in implementation.
4. Deployment.
5. Implementation for other Grid MW.
6. Further investigations of other Grid API Interfaces.

The following paragraphs contain information about our research and development plans.

5.1 Local PROOF cluster

Before we can seriously look for the solution, which could help us to use PROOF on the Grid, we need to get experience with PROOF. This is the reason why we need to have a local PROOF cluster first, from a most resent ROOT version (we require the most resent version of the PROOF, because it is actively developing package of the ROOT).

The GSI GridTeam had already pretty much worked with PROOF [14], also in this terms GSI GridTeam led a course in the GridKa School 2005 “Hands On ROOT / PROOF” [16].
This is a good and valuable experience, but since the last time our team worked on PROOF a lot of changes have been made by ROOT team in the PROOF package. We therefore require fresh and recent PROOF cluster installed and investigated in all aspects. We should find out a suitable solution of using PROOF algorithms in gLite Grid.

5.2 "Marriage" of the PROOF and gLite

The result of the research of the local PROOF cluster (see §5.1) is a part of the solution which could help us to marry PROOF and gLite. We need the solution, which could give us the same possibilities as in case of AliEn and PROOF (see §3.1.2).

TODO: Describe Interactive gLite job.
TODO: # Develop a schema, which could help to process PROOF jobs on gLite MW

5.3 Complete GAW and RGlite plug-in implementation

As it was already mentioned RGlite plug-in have currently all required functionality to be a ROOT Grid Interface plug-in for the gLite Grid middleware, it implements all of the methods of ROOT Grid Interface. Also it was stated that RGLite is completely based on GAW (GSI GridTeam gLite API wrapper library).

In spite of the fact that all of the functionality, required by ROOT Grid Interface, is already implemented we see some room for improvements and at least one missing functionality, we would like to implement.
To the next stage of the project we are planning the following implementation improvements in the RGLite and in GAW:

- GAW Job Manager is going to be slightly redesigned and will be implemented as a C++ template class in order to support type traits and several different Job Managers of gLite (for example WMSUI and WMProxy).
- Interface of the GAW Core Manager is going to be change accordingly to GAW Job Manager.
- GAW Job Manager will get better support of different job types (such as interactive jobs) implemented.
- BOOST [17] tests will be used instead of “home-made” tests in order to bring quality assurance of the product to higher level.
- Several methods of RGLite plug-in are going to be improved in terms of usability (internal code improvements, better logging and user interaction). Especially this will affect *TGLite* interface.
- There are currently several test cases (implemented as ROOT macros) for RGLite and those can be used as examples for end-users. However it requires more complex usage cases and examples. This will help users to faster understand and get to know this interface.
- Project and code documentation are going to be constantly revised.

TODO: Write something about development of small tools (for example, a tool which will scan gLite UI and generate GAW config. automatically.)

5.4 Deployment

As soon as we get first fully operational alpha version of the product (for all of the means – the whole implementation of the model for gLite Grid) we will try to find test users. Also, starting from now November 2006 there will be place from were RGLite plug-in, GAW library could be download so that users who may want to play with could have there chances. RGLite can give some advantages already now, because it allows everyone who uses ROOT to design and implement Grid enabled application, but the real power for data analysis will come as soon as we'll implement complete a model and will get all of the components (RGlite, ROOT, PROOF, gLite) operation together.

Deployment stage of the project will consists of three major parts:

1. Alpha. On this stage we would like to intensively test our schema in "real life" analysis and including stress testing. We are expecting to get some individual users or small group of users to have close contact while they use our solution. Most probably here will be some design corrections and major improvements done. This stage is all about users, it is very important to get some test users and a feed back from them in order to improve the solution accordingly. There will be Issue tracker installed (like Bugzilla) where users can register an issue or bug and get a feedback from developers.
2. Beta (or pre production). Here we going to massively advertise our product and try to deploy the system to bigger collaborations or groups of users. This is a very difficult task. Therefore every possibility must used, like conferences, workshops or locale presentations. A regularly updated, dedicated product web site with available downloads and documentation should be launched.
3. Production. This stage is a long term plan and has in mind the usage of the solution in a production system. The strategy for the production system is going to be designed on later stages of the project.

To bring the solution to the Alpha stage is a current plan for the deployment.

5.5 Implementation for other Grid MW

When our system will get Alpha release, we will start to develop implementations for other Grid middleware, for example Globus4. This means that we would need to implement a ROOT plug-in for Globus4 and research and implement mechanism to enable PROOF job summation in Globus4 environment.
Implementation for different middleware will improve the system and mostly it will improve ROOT Grid Interface. While implementing plug-in for Globus4 we will see is there any inconsistency ion the interface or some possible improvements as we did so while were implementing plug-in for gLite middleware.

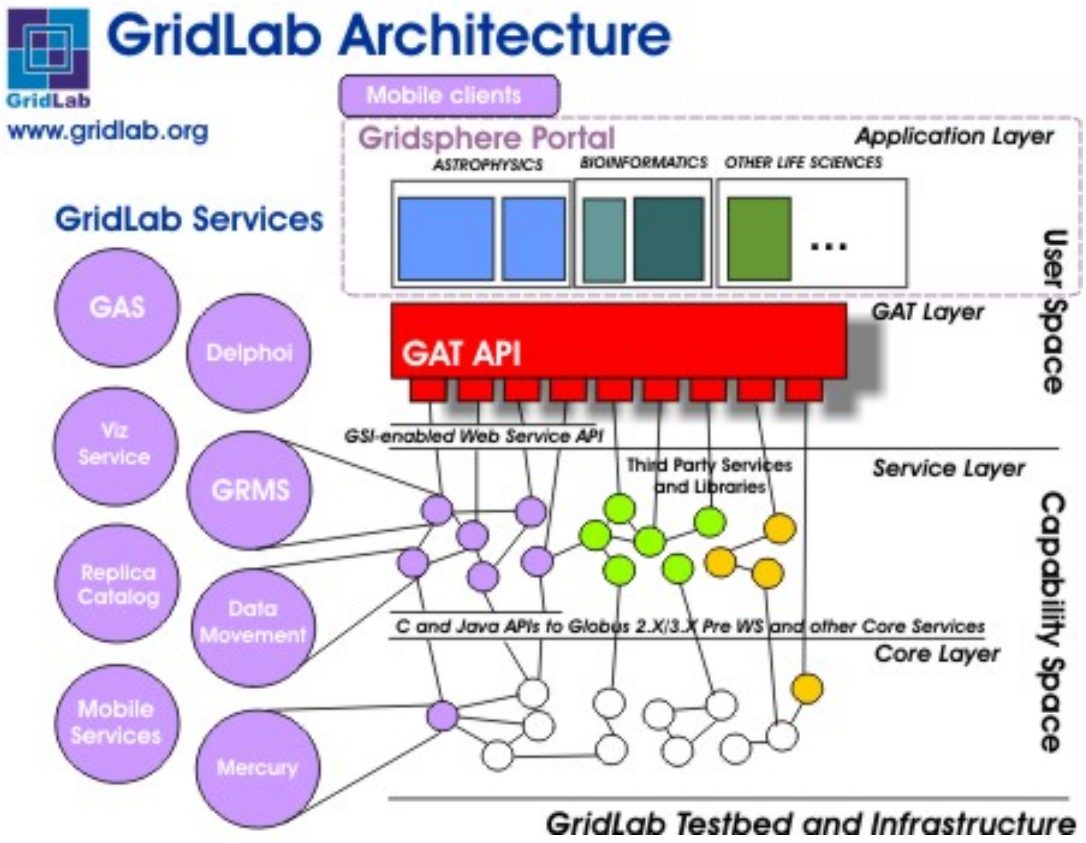
5.6 Investigate other Grid API Interfaces

We realize that it is important to be well informed about modern Grid technologies. Grid is a very young technology and it grows pretty fast. We therefore should keep up-to date our knowledge about generic Grid APIs, modern Grid technologies and new or production Grids.

As a short term plan for the next stage of the project we assigned investigation and tests of the following generic Grid APIs: Grid (Lab) Grid Application Toolkit [12] and Simple API for Grid Apps (SAGA) [13].

5.6.1 GAT

"GAT is a set of coordinated, generic and flexible APIs for accessing Grid services from e.g. generic application codes, portals, data managements systems, together with working implementations provided by the tools developed in the Grid Lab project (See the figure below). GAT is designed in a modular plug-and-play manner; such that tools developed anywhere can be plugged into GAT.



As shown in the above figure, GAT and the GAT API sit between Grid applications and numerous types of grid middleware. GAT lifts the burden of grid application programmers by providing them with a uniform interface to numerous types of grid middleware. As a result,

grid application programmers need only learn a single API, that of GAT, to obtain access to the entire grid." [12]

5.6.2 SAGA

"The SAGA Research Group at GGF (Global Grid Forum) strives to define a high level API for developers of Grid Applications. Instead of interfacing directly to Grid Services, the applications can so access basic Grid Capabilities with a simple, consistent and stable API. For example, to copy a file on a Grid, not much more than

```
call fileCopy (source, destination);
```

should be needed. Although this example is simplified, it illustrates the motivation for our work. The APIs specified by this WG will deliver a similar level of abstraction for several sets of basic Grid operations. The precise set of operations is yet to be defined, and bases on Use Cases elicited by the group, but our initial focus will be on file transfer and job submission.

The group will lower the barrier for scientific application developers to make use of the grid by providing a small, consistent API for the operations of interest, the Simple API for Grid Applications (SAGA)." [18]

6 References

[1] „Alice Grid Activities“, Dr. Peter Malzacher (GSI), Seminar Datenverarbeitung in der Hochenergiephysik DESY, 2006-05-27

[2] AliEn Home site, http://alien.cern.ch/twiki/bin/view/AliEn/Home

[3] gLite middleware, http://glite.web.cern.ch/glite/

[4] ROOT, http://root.cern.ch/

[5] AliEn, http://alien.cern.ch/twiki/bin/view/AliEn/Home

[6] PROOF, http://root.cern.ch/root/PROOF.html

[7] CERN Savannah, https://savannah.cern.ch/

[8] RGlite and GAW documentation, http://wiki.gsi.de/cgi-bin/view/Grid/RGLiteAndGAW

[9] GSI’s D-Grid Wiki, http://wiki.gsi.de/cgi-bin/view/Grid/WebHome#D_Grid_Initiative

[10] WMProxy API, http://trinity.datamat.it/projects/EGEE/wiki/wiki.php?n=WMProxyAPI.APIDocumentation

[11] OLD-problematic JRA1: Data Management site, http://cern.ch/egee-jra1-dm/, http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/doc.htm

[12] Grid (Lab) Grid Application Toolkit, http://www.gridlab.org/WorkPackages/wp-1/

[13] “Simple API for Grid Applications (SAGA)”, Thilo Kielmann (Vrije Universiteit, Amsterdam), http://www.ggf.org/GGF17/materials/319/kielman-ggf17.pdf

[14] GSI’s PROOF activity, http://wiki.gsi.de/cgi-bin/view/Grid/TheParallelRootFacility

[16] GSI’s PROOF activity, GridKa School 2005 “Hands On ROOT / PROOF”, http://gks05.fzk.de/

[17] Boost C++ Libraries, http://www.boost.org/

[18] Simple API for Grid Applications (SAGA), http://wiki.cct.lsu.edu/saga/space/start

[19] Grid source forge, “Project: SAGA-RG”, https://forge.gridforum.org/projects/saga-rg/