THE ZEUS GRID-TOOLKIT – AN EXPERIMENT INDEPENDENT LAYER TO ACCESS GRID SERVICES

K. Wrona, H. Stadie, M. Ernst, R. Mankel, DESY, Hamburg, Germany
J. Ferrando, University of Glasgow, Glasgow, United Kingdom

Abstract

The HERA luminosity upgrade and enhancements of the detector have led to considerably increased demands on computing resources for the ZEUS experiment. In order to meet these higher requirements, the ZEUS computing model has been extended to support computations in the Grid environment. We show how to use the Grid services in the production system of a real experiment and point out the main issues, which must be addressed in order to use the Grid resources routinely and efficiently. We present the ZEUS Grid-toolkit designed as an additional layer between the Grid and experiment specific software. It provides a general interface for job management and data handling, which makes our application software independent of the actual Grid middleware software version. Different Grid middleware implementations as LCG/EGEE [1] or Grid3/OSG [2] may be used simultaneously and smooth migration is possible as new middleware implementations appear (e.g. gLite [3]). The job efficiency is significantly improved by introducing fault tolerant methods. The toolkit provides extensible Perl [4] classes for job management and implements additional features like dynamic creation of job description, automatic job resubmission and validation of the job results. The toolkit has been successfully used in the integrated ZEUS Monte Carlo production system for more than a year.

INTRODUCTION

The ZEUS experiment at HERA collider is one of the first currently data-taking projects, which successfully deployed a Grid-based Monte Carlo simulation suite at the mass production scale. The system has been routinely used in the daily physics analysis for more than a year.

After the luminosity upgrade of HERA accelerator and enhancements of the ZEUS detector, the demands for Monte Carlo production have increased considerably. ZEUS experiment had already been using a distributed MC production system since early 90’s. The system had used dedicated farms spread all over the world. Further extension was possible only through the emerging Grid infrastructure. Extending the ZEUS computation model to support simulation in the Grid environment became indispensable in order to meet higher requirements.

Efficient usage of available Grid resources was an important issue right from the beginning. Therefore a careful design was mandatory. Legacy software used in the standard system provided the basis for the integrated Monte Carlo production system, but the Grid services had to be interfaced in the transparent way.

THE ZEUS GRID-TOOLKIT

Motivation

The grid services and tools delivered to the users through different Grid technologies are still not fully mature. Different Grid middleware flavours provide client tools with different interfaces. Furthermore, the software is under heavy development and interfaces keep changing between releases. Various services are being re-implemented, which frequently leads to the enforced modifications of the users software. Many features useful for the job and data management are still missing or the existing implementation is not satisfactory. For example, there are no tools to manage users’ jobs on a large scale. On the other side, over the time many services get gradually improved and therefore it is important that the users benefit from them as soon as they come available. In order to improve the management of the project-specific software and allow for quick adaptation to the frequent modifications related to the Grid services, the ZEUS Grid-Toolkit has been developed.

General concept

The ZEUS Grid-Toolkit is a software layer (Figure 1), which provides an abstract interface to the project specific software. It encapsulates all details related to the particular Grid middleware implementation. Object-oriented design techniques have been employed in order to make the toolkit maintainable over a longer time scale. Different Grid technologies may be used simultaneously and migrations become straightforward.

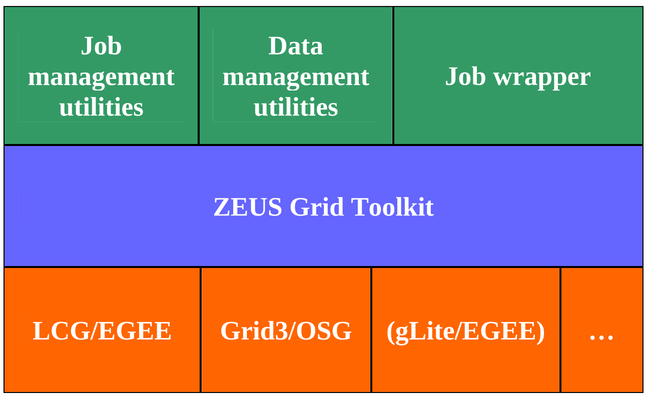


Figure 1: The layout of the ZEUS Grid-Toolkit

Toolkit functionality

The ZEUS Grid-Toolkit provides a set of Perl classes (modules) for the job management on the User Interface. In contrary to the LCG client tools it is possible to store on the client host not only the job identification string, but the whole set of parameters related to the Grid job. After the job is submitted to the Grid, the corresponding object of the Grid::Job class is serialized and can be stored in a local database for further usage. The toolkit classes provide methods to submit jobs, query their status, retrieve the output, validate the results of the jobs and if necessary resubmit the job again. In addition the project specific information can be easily added to the job object, by extending the class through the inheritance.

Another set of classes supports the data management utilities. Tools built on top of these classes handle the registration of files in the catalog, transfer of the files between different sites and validation of the transferred files. In case of failure, the operation may be repeated a configurable number of times until it succeed.

A timeout can be enforced for execution of any external program. This proved to be very useful, in particular while running batch jobs where the user does not have direct control on the execution environment. The logging facility and the verification framework for the job results significantly improve the error detection and recovery from failures.

The toolkit contains also a set of general-purpose classes, which help developers to write maintainable code. This includes basic structure classes like Array, Hash, Iterator as well as the command line parser utilities, classes handling the configuration parameters or providing persistency for the objects.

In the following the design of the most important classes is explained.

DESING AND IMPLEMENTATION OF THE TOOLKIT

Job management classes

In Figure 2 a simplified UML class diagram for the job management classes is shown. By using the "Strategy pattern" the interface is clearly separated from the implementation classes. The abstract class *Grid::JobCommand* declares methods, which must be defined by concrete implementation classes. These implementation classes are mostly wrappers around the client tools provided by particular Grid middleware. At the moment, two implementations are supported. One uses tools delivered by LCG/EGEE middleware while the second one uses the Globus [5] toolkit. Whenever new and more attractive implementations appear the software may be easily adjusted to use its methods. For example, the gLite middleware would be a natural candidate.

The user script does not need to deal with these different approaches, because the code uses only the

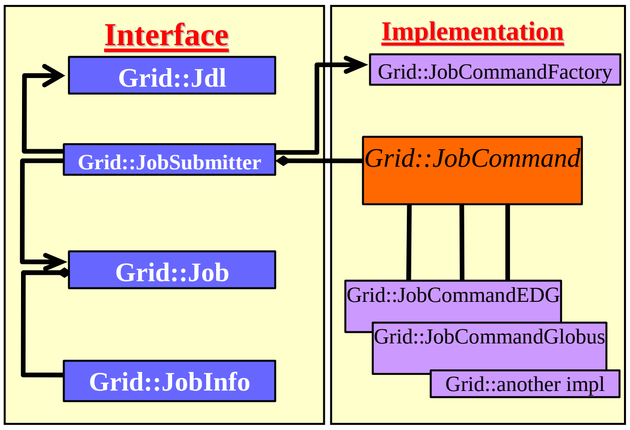


Figure 2: Simplified UML class diagram for job management classes

interface classes. In all cases the job is defined through the Grid::Jdl class interface. Upon the job submission to the LCG site, the description of the job is converted to a JDL file, while in case of Grid3/OSG site the globus submission tool is run with proper options.

The Grid::Jdl class provides also methods, which restrict the job submission to those sites which properly handle the user requests. For example, a site with temporary problems may accept Grid jobs, but never submit them to its local batch system. The user can supply a file with the "white list" of acceptable Computing Elements.

The Grid::JobCommandFactory ensures that the proper implementation is used when new jobs are submitted. The decision is based on the availability of installed packages and the user configuration file:

DefaultImplementation LCG
DefaultTimeOut 180
GlobusStorageArea dcache.desy.de/pnfs/zeusmc/spool

Once the user submits the job using the Grid::JobSubmitter class, the corresponding job object is created. The run-time object may be serialized and stored in a database. The persistency feature is provided by the ZEUS::Storable class. The object can be retrieved from the database and instantiated at any later time.

The Grid::Job class may be also enriched through the class' inheritance. New methods can be defined according to the needs of the project. The toolkit contains an example of the code, which shows how to display a history of the job, validate the job output and re-submit failed jobs.

Data management classes

One of the most important tasks users have to handle when running jobs on the Grid is the management of data. This task involves many Grid services like catalogs, space management, replication of data sets, and data transfer. Existing implementations of these services however, are not yet on a satisfactory level and very often simple operations fail due to temporary problems.

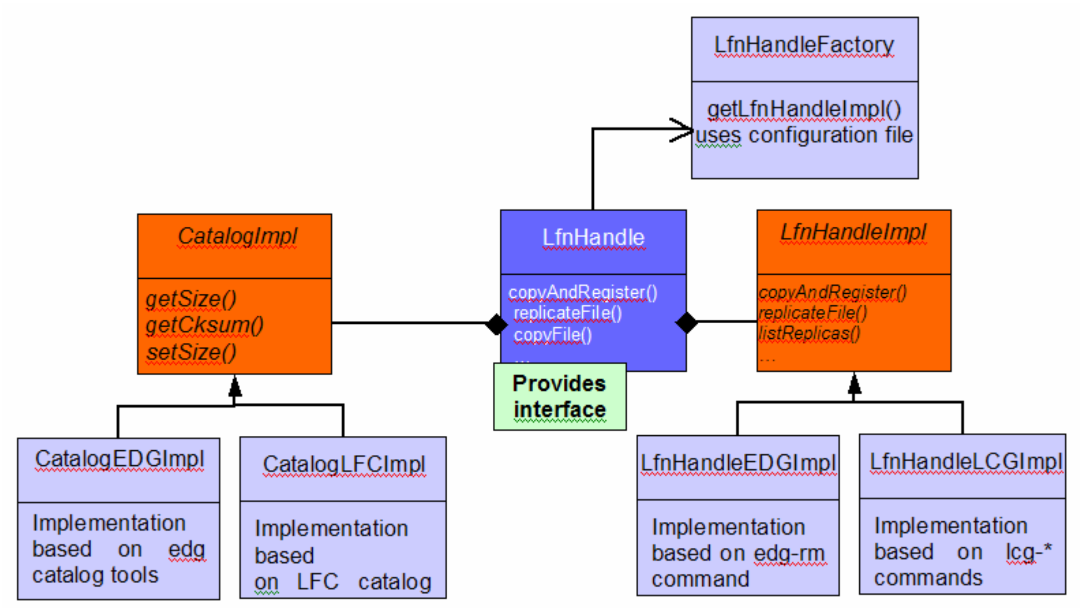


Figure 3: UML class diagram for data management

The ZEUS Grid-Toolkit strives to overcome these failures by implementing appropriate recovery procedures. In case recovery is impossible, the user is informed. The most undesirable effect users have to experience is the lack of any response from a Grid service. Unfortunately, this may still happen frequently. Therefore, the toolkit implements configurable timeout enforcement for all data-related operations. Depending on the task, the user can choose an appropriate timeout value, and in addition the operation can be retried several times before it is regarded to be unsuccessful. Below an example of a configuration file is shown:

```
VO ZEUS
DefaultImplementation LCG
DefaultSE srm-dcache.desy.de
CatalogImplementation LFC
CatalogHost grid-cat2.desy.de
DefaultTimeout 1200
DefaultMaxTries  3
DESY_DE_MaxTries  2
```

Apart from the earlier mentioned timeout and retry configuration parameters, the list also contains parameters, which have to be delivered as options to most of the tools. Examples are the virtual organization name, the catalog host or the storage element.

The data management classes (Figure 3) are organized in a way similar to the classes handling users' jobs. The interface is provided by one class, LfnHandle, while all details are hidden from the user in the set of implementation classes. There are two internal independent implementations: one for the catalog operations and the second for the data transfer. The catalog is used to provide a global namespace for registered files. The class handles registration of files. Apart from the logical file name, additional parameters are stored in the catalog. These are: owner of the file, file size, time of creation and a control sum calculated over the content of the file. These parameters are calculated during the registration step and then are used to check the consistency of the file, for example after the replication to another storage element. Both EDG and LFC catalogs are currently supported. User can choose which one should be used by setting the configuration parameter.

The transfer of files between different hosts on the Grid is implemented using the tools delivered by the LCG middleware. At an earlier stage, the edg-rm command line tool was encapsulated in the implementation class. Later on, the new set of commands became available. These new commands were wrapped into the second implementation class. This approach allowed us to use both command sets simultaneously during the migration phase.

Other toolkit classes

The toolkit contains also classes for more general purposes. These include code for flexible logging utilities, command line parsing, execution of external programs, filesystem operations, time manipulation, and basic structures like Array, Hash, or Vector.

For example the class ZEUS::ShellEnvSystem provides a method to call any program from the Perl script. This class has a built-in mechanism, which enforces a configurable timeout if user needs it. It is also implemented in a way that both standard output and return code are accessible, which is not provided by the standard Perl modules. Other classes like ZEUS::Storable, which provides a persistency mechanism for Perl objects, or command line parser, are just wrappers around the standard Perl modules. These wrappers are mainly useful for developers who want to keep their code clear and easily maintainable.

THE MONTE CARLO PRODUCTION SYSTEM

The ZEUS Grid-Toolkit has been used for the first time in the Integrated Monte Carlo Production System for ZEUS experiment. The original distributed system was extended to support computation in the Grid environment. Detailed description of the system is given in the [6]. In Figure 4 the architecture of the system is shown.

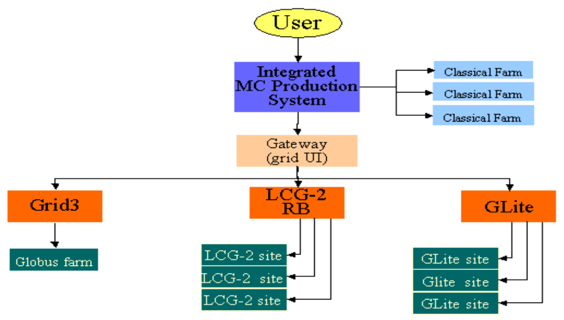
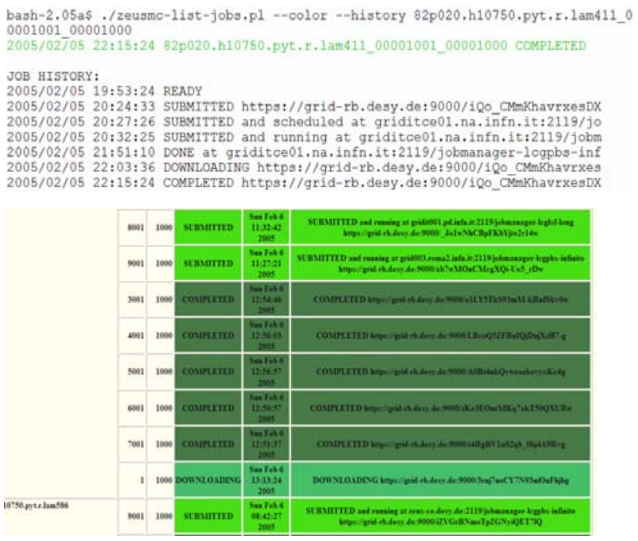


Figure 4: ZEUS MC production system

An original user request submitted to the Integrated MC Production System may be run on the classical farms or on the Grid. The requests assigned to the Grid are sent to the User Interface (UI) host, which acts as a gateway to the Grid. Each request is divided into a number of independent tasks. These tasks are dynamically translated into the Grid jobs using the Grid::Jdl class. The jobs are then submitted to the Grid, and Perl objects representing these jobs are serialized and stored in the MySQL [7] database. The Grid::Job class is extended through the inheritance mechanism. Additional parameters strictly related to the MC request are kept in the objects. Also the full Grid::Jdl object is stored in Grid::Job, which allows the system automatically resubmit failed job. In the production system, both LCG/EGEE and Grid3/OSG middleware flavours are used. Job objects can be accessed from the database and de-serialized to the fully functional runtime objects. The object-oriented approach ensures that the methods, which are used to query the job status or retrieve the output, use the proper implementation classes. By using a common interface, the Gateway software which is by itself complex does not have to know anything about the tools delivered by a particular middleware. This separation turned out to be highly beneficial. Any changes triggered by the development of Grid software are thus localized and restricted to the ZEUS Grid-Toolkit. Migration can be done very fast between subsequent releases.

Monitoring of the jobs is crucial for a mass production system. The objects stored in the database serve as a primary source of information about the status of the production. The full history of a submitted task can be inspected using the command line and web interface (Figure 5).



```
bash-2.05a$ ./zeusmc-list-jobs.pl --color --history 82p020.h10750.pyt.r.lam411_0
0001001_00001000
2005/02/05 22:15:24 82p020.h10750.pyt.r.lam411_00001001_00001000 COMPLETED

JOB HISTORY:
2005/02/05 19:53:24 READY
2005/02/05 20:24:33 SUBMITTED https://grid-rb.desy.de:9000/iQo_CMmKhavrxesDX
2005/02/05 20:27:26 SUBMITTED and scheduled at griditce01.na.infn.it:2119/jo
2005/02/05 20:32:25 SUBMITTED and running at griditce01.na.infn.it:2119/jobm
2005/02/05 21:51:10 DONE at griditce01.na.infn.it:2119/jobmanager-lcgpbs-inf
2005/02/05 22:03:36 DOWNLOADING https://grid-rb.desy.de:9000/iQo_CMmKhavrxes
2005/02/05 22:15:24 COMPLETED https://grid-rb.desy.de:9000/iQo_CMmKhavrxesDX
```

	8001	1000	SUBMITTED	Sun Feb 6 11:32:42 2005	SUBMITTED and running at gridit001.pd.infn.it:2119/jobmanager-lcglsf-long https://grid-rb.desy.de:9000/_Jo1wNhCBpFKbYjte2r14w
	9001	1000	SUBMITTED	Sun Feb 6 11:27:21 2005	SUBMITTED and running at grid003.roma2.infn.it:2119/jobmanager-lcgpbs-infinite https://grid-rb.desy.de:9000/zh7wMOuCMcgXQ6-Ue5_rDw
	3001	1000	COMPLETED	Sun Feb 6 12:54:46 2005	COMPLETED https://grid-rb.desy.de:9000/[illegible]
	4001	1000	COMPLETED	Sun Feb 6 12:56:03 2005	COMPLETED https://grid-rb.desy.de:9000/[illegible]
	5001	1000	COMPLETED	Sun Feb 6 12:56:57 2005	COMPLETED https://grid-rb.desy.de:9000/[illegible]
	6001	1000	COMPLETED	Sun Feb 6 12:50:57 2005	COMPLETED https://grid-rb.desy.de:9000/[illegible]
	7001	1000	COMPLETED	Sun Feb 6 12:51:37 2005	COMPLETED https://grid-rb.desy.de:9000/[illegible]
	1	1000	DOWNLOADING	Sun Feb 6 13:13:24 2005	DOWNLOADING https://grid-rb.desy.de:9000/[illegible]
10750.pyt.r.lam586	9001	1000	SUBMITTED	Sun Feb 6 08:42:27 2005	SUBMITTED and running at zeus-ce.desy.de:2119/jobmanager-lcgpbs-infinite https://grid-rb.desy.de:9000/[illegible]

Figure 5: Job monitoring tools

SUMMARY

The ZEUS Grid-Toolkit has been designed and developed in order to efficiently use the existing client tools for accessing Grid services. Careful object-oriented design has led to the solution, which separates the project specific code and the Grid tools delivered by concrete middleware implementation or its subsequent version. The differences between Grid middleware flavours are encapsulated in the implementation classes while the interface is independent from it. The toolkit fixes known deficiencies of contemporary Grid software and extends its functionality. Simultaneous usage of several Grid technologies is possible. Due to the usage of the presented toolkit, we have been able to increase the efficiency of the ZEUS Monte Carlo jobs significantly.

ACKNOWLEDGMENTS

The authors of this article would like to thank all Grid sites representatives who support the ZEUS VO and allow us using their computing resources, Grid farms administrators for the cooperation in solving technical issues.

REFERENCES

[1] LCG: http://lcg.web.cern.ch/LCG/
[2] Open Science Grid: http://www.opensciencegrid.org/
[3] gLite: http://glite.web.cern.ch/glite/
[4] Perl: http://www.perl.com/
[5] I. Foster, C. Kasselmann "Globus: A Metacomputing Infrastructure Toolkit" Intl. J. Supercomputer Applications 11(2) (1997) 115-128
[6] H.Stadie at al., "Long-term Experience with Grid-based Monte Carlo Mass Production for the ZEUS Experiment"
[7] MySQL: http://www.mysql.com/