HappyFace as a monitoring tool for the ATLAS experiment

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
„Doctor rerum naturalium“
der Georg-August-Universität Göttingen

im Promotionsprogramm ProPhys
der Georg-August University School of Science (GAUSS)

vorgelegt von

Haykuhi Musheghyan

aus Eriwan, Armenien

Göttingen, 2016

Betreuungsausschuss

Prof. Dr. Arnulf Quadt
II. Physikalisches Institut, Georg-August-Universität Göttingen

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen - GWDG

Mitglieder der Prüfungskommission:

Referent:	Prof. Dr. Arnulf Quadt II. Physikalisches Institut, Georg-August-Universität Göttingen
Koreferent:	Prof. Dr. Ramin Yahyapour Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen - GWDG

Weitere Mitglieder der Prüfungskommission:

Prof. Dr. Jens Grabowski
Institute of Computer Science, Georg-August-Universität Göttingen

Prof. Dr. Wolfram Kollatschny
Institut für Astrophysik, Georg-August-Universität Göttingen

Prof. Dr. Stan Lai
II. Physikalisches Institut, Georg-August-Universität Göttingen

Jun.-Prof. Dr. Steffen Schumann
II. Physikalisches Institut, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 05.08.2016

Referenz: II.Physik-UniGö-Diss-2016/04

A dream doesn’t become reality through magic; it takes sweat, determination and hard work.

Colin Powell

Abstract

The importance of monitoring on HEP grid computing systems is growing due to a significant increase in their complexity. Computer scientists and administrators have been studying and building effective ways to gather information on and clarify a status of each local grid infrastructure. The HappyFace project aims at making the above-mentioned workflow possible. It aggregates, processes and stores the information and the status of different HEP monitoring resources into the common database of HappyFace. The system displays the information and the status through a single interface.

However, this model of HappyFace relied on the monitoring resources which are always under development in the HEP experiments. Consequently, HappyFace needed to have direct access methods to the grid application and grid service layers in the different HEP grid systems. To cope with this issue, we use a reliable HEP software repository, the CernVM File System. We propose a new implementation and an architecture of HappyFace, the so-called grid-enabled HappyFace. It allows its basic framework to connect directly to the grid user applications and the grid collective services, without involving the monitoring resources in the HEP grid systems.

This approach gives HappyFace several advantages: Portability, to provide an independent and generic monitoring system among the HEP grid systems; Functionality, to allow users to perform various diagnostic tools in the individual HEP grid systems and grid sites; Flexibility, to make HappyFace beneficial and open for the various distributed grid computing environments. Different grid-enabled modules, to show all current datasets on a given storage disk and to check the performance of grid transfers among the grid sites, have been implemented. The new HappyFace system has been successfully integrated and now it displays the information and the status of both the monitoring resources and the direct access to the grid user applications and the grid collective services.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The Worldwide LHC Computing Grid (WLCG) project is providing the computing resources for processing and analysing the data gathered by the LHC experiments. WLCG is based on a concept of grid and has a very complex infrastructure. This complexity requires the usage of various monitoring and meta-monitoring tools, which is a critical and non-trivial task. In order to deal with this task, several monitoring tools had been designed and implemented over the years by computing models of different experiments. My thesis is related to one of these experiments, which is ATLAS. In ATLAS, besides the main tools that are described in chapter 4, there are a number of other auxiliary tools. All these tools together assist to solve the challenging task of scalability and reliability of the distributed services at more than 170 sites.

One of the auxiliary tools in ATLAS is HappyFace (see Chapter 6). Initially, HappyFace was designed as a meta-monitoring tool that aggregates, processes and stores information from various monitoring tools into a single web interface. It has a modular structure where each module is responsible for extracting, processing and storing data from the certain WLCG monitoring sources. Being a meta-monitoring tool, HappyFace was strongly dependent on the WLCG monitoring sources.

During the LHC Run 1 phase, the ATLAS computing model was stable and some minor changes that were applied to the model were not really harming the modules' work of HappyFace. During this phase, HappyFace had an essential progress in terms of source code optimization and improvement, module development and usage by other grid sites.

However, the data growth during the LHC Run 2 phase brought major and fundamental changes to the ATLAS computing model. All ATLAS tools were tuned. Some of them were completely redesigned and some had changes in their initial models by adding or removing functional components. But changes in the ATLAS computing model could not pass by HappyFace without affecting it and as a result, its stability became highly labile. The dependency on monitoring sources became a clear weak point of HappyFace. Moreover, new features appeared that also had to be implemented in HappyFace.

Raised problems during the LHC Run 2 phase:

- **Make HappyFace a stable system again**
- **Reduce manual interactions of site administrators**

Raised problems required a certain solution. Thus, the goal of this thesis as well as my contribution is to provide a general solution for these problems in HappyFace.

1. How to make HappyFace again stable ?

To make HappyFace a stable system again was a challenging task taking the current situation of ATLAS computing model into account. The dependency on the WLCG monitoring sources was the weak point of HappyFace. Therefore, by solving the problem of dependency on the WLCG monitoring sources, HappyFace will become again stable.

To solve the problem, I decided to modify the initial model of HappyFace by designing and implementing a new extension for it, the so-called grid-enabled HappyFace. By this newly designed extension, HappyFace got access to the grid and thus solved the problem (see Section 6.3).

2. How to reduce the manual interactions of site administrators ?

Using the existing ATLAS software alone is not enough to check the site performance, manual interactions are required. Therefore, three different grid-enabled modules (GridFTP, GridSRM, ATLAS DDM) were designed and developed (see Section 6.3.3). The existence of these modules solves the problem of manual interactions in terms of performing file transfers among sites and listing all current datasets for the given storage disk or a storage pool.

The new extension opens a new stage of development for HappyFace. Within this extension, still a number of new modules can be easily developed and implemented for different purposes in HappyFace.

An overview of the structure of thesis chapters are provided below.

The purpose of chapter 1 is to provide a basic information about CERN, LHC experiments, physics and the explanation of the need of such a complex infrastructure as WLCG computing infrastructure. Without WLCG computing infrastructure and, in particular, without the ATLAS computing model storing, distributing, analising experimental data would be impossible. WLCG infrastructure is based on the concept of grid computing, therefore chapters 2 and 3 are describing the concept of grid computing and grid middleware. In particular, chapter 2 includes all necessary information about the security of grid systems, description of storage systems, computing elements and workload management systems.

As in ATLAS we are using gLite middleware, consequently in chapter 3 the main gLite components are described. This chapter also contains a detailed information about GoeGrid. Chapter 4 is describing the ATLAS computing model by providing information about the main ATLAS software. The purpose of this chapter is to show the complexity of the ATLAS computing model and to present the actual tools work. Chapter 5 is describing the importance of using monitoring and meta-monitoring tools.

All these mentioned above chapters are needed to understand what we have and to start with the actual research topic, which is covered in chapter 6. Chapter 6 is providing a detailed description about HappyFace and its new extension. It contains a detailed explanation about the design and implementation of new extension as well as the module development withing this new extension. The conclusion and appendices part are covered by chapters 7 and appendix A and B. In appendix B is a step by step description of the installation of the grid environment and the HappyFace instance.

1.2 LHC

The Large Hadron Collider (LHC) [1] is the world's largest, most powerful and the most complex particle accelerator. The LHC at CERN promises a major step forward in understanding the fundamental nature of matter. The LHC consists of a 27-kilometre ring of superconducting magnets with a number of accelerating structures to enhance the energy of the particles along the way (Figure 1.1).

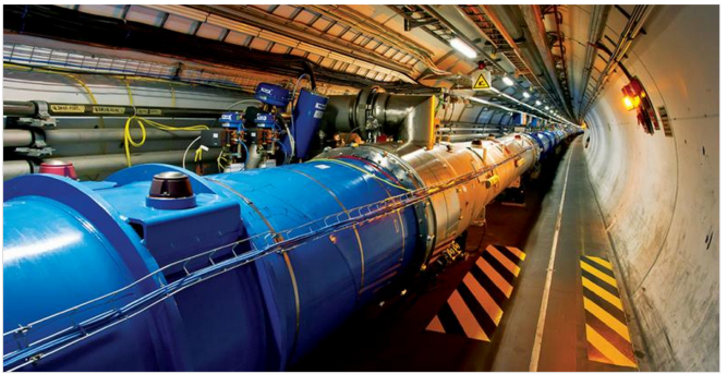


Figure 1.1: The Large Hadron Collider (LHC).

It started up on the 10th of September 2008 with over 10,000 scientists and engineers from over 100 countries, as well as hundreds of participating universities and laboratories.

The first accelerator run operated between March 2010 and February 2013 with an initial 3.5 teraelectronvolts (TeV) per beam (total center of mass energy of 7 TeV). In 2012, the center of mass energy increased up to 4 TeV per beam (total center of mass energy of 8 TeV). In 2015, LHC Run 2 [2] was restarted with a proton-proton collisions at the center of mass energy of 13 TeV. An energy unit used in particle physics is teraelectronvolts (TeV), see Equation 1.1.

$$1\text{TeV} = 10^{12}\ \text{electronVolts(eV)} \tag{1.1}$$

This new stage allows the LHC experiments to study or explore the nature and physics laws at higher energies.

In February 2013, the LHC's first run officially ended, and it was shut down for planned upgrades. "Test" collisions were restarted in the upgraded collider in April 2015, reaching 6.5 TeV per beam in May 2015 (13 TeV in total, the current world record for particle collisions). Its second research run began on schedule, on 3 June 2015.

The LHC's aim is to allow physicists to analyse the predictions of different theories of particle and high-energy physics, in order to prove or disprove the existence of the theorised Higgs boson [3] and the large family of new particles predicted by supersymmetric theories [4].

The Standard Model (SM) [5], [6], [7] (see also Section 1.3) of particle physics is a theory which was developed in the mid-1970s. describes the fundamental particles and their interactions. It predicted a wide variety of phenomena and explains the majority of experimental results. However, the SM is not complete. There are still many open questions and the LHC will help to answer these questions (see Section 1.7).

Seven different experiments have been designed and constructed for the LHC for certain kinds of research. There are four main experiments: ATLAS, CMS, LHCb, ALICE. The biggest of these experiments are ATLAS and CMS, whose purpose is to investigate the largest range of physical phenomena within and beyond the SM. However, they both offer different technical solutions.

ALICE and LHCb are specialised in a specific physical phenomenon. ALICE is focused on studying the heavy-ion collisions. Here, the energy densities are extremely high in order to produce forms of quark-gluon plasma [11]. In the phase of forming a quark-gluon plasma, the energy is so high that it leads to the melting of protons and neutrons. As a result, the quarks lose their tight connection with the gluons.

LHCb is specialised in studying the "beauty quarks" or "b quarks" and their properties. The study meant to explain the differences between matter and antimatter.

These four experiment detectors were built underground (100 m below ground) in huge caverns on the LHC ring (Figure 1.2).

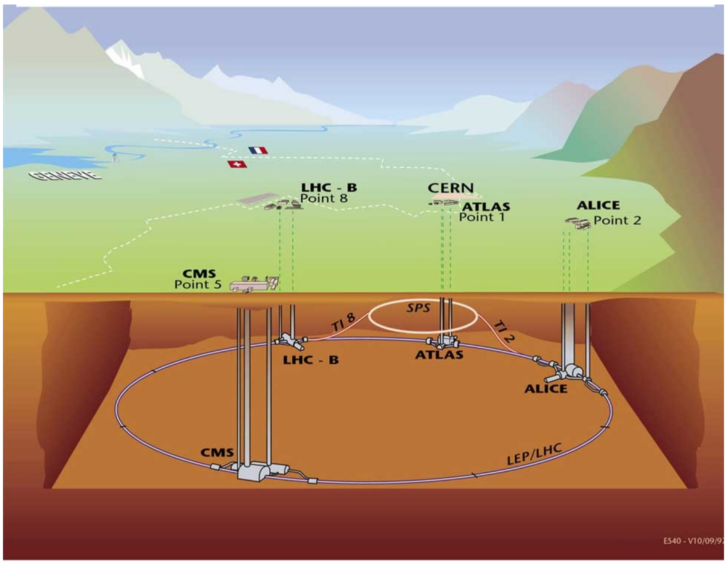


Figure 1.2: Experiments running at the LHC.

The LHC Computing Grid is an important part of LHC and is the biggest computing grid in the world. It was designed at CERN (see Section 1.2.1) to manage a significant amount of data coming from different experiments. The Worldwide LHC Computing Grid (WLCG) [13] is a global collaboration of a number of computing centers around the world that are connected to each other with the grid infrastructure. More details about WLCG is described in Section 1.9.

1.2.1 CERN

The European Organisation for Nuclear Research known as CERN (derived from the name "Conseil Européen pour la Recherche Nucléaire") is a European research organisation that operates the largest particle physics laboratory in the world. It was established in 1954 and is located in the northwest suburb of Geneva on the Franco–Swiss border.

Particle accelerators and detectors are built at CERN, which are needed for the high-energy physics research. Accelerators enhance beam energies of particles to high energies before beams collide with each other or with stationary targets. Detectors observe and record the results of these collisions. CERN has made several important achievements in particle physics through the different experiments. These achievements are collected in a book with the name "Prestigious Discoveries at CERN" [14].

1.3 The Standard Model of Particle Physics

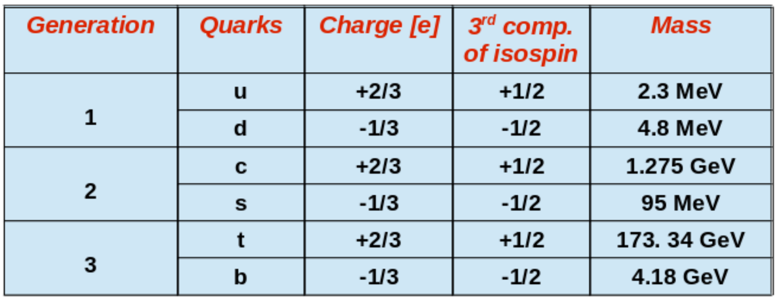
The *Standard Model (SM)* [5], [6], [7] is the theory that describes the fundamental particles and interactions between them. The SM is a theory related to the electromagnetic, strong and weak nuclear interaction force and the classification of all the known subatomic particles. The SM represents current understanding about Nature and its interactions and explains the phenomena in Particle Physics. The weak and electromagnetic interactions or the so-called electroweak force are related to Quantum Electrodynamics (QED) [8] theory and the strong interactions are related to Quantum Chromodynamics (QCD) [9] theory.

The SM includes two types of particles: *fermions* and *bosons*. The *fermions* have an intrinsic angular momentum called *spin*, equal to half-integer. *Bosons* called the particles that have an integer spin. Each particle has its corresponding antiparticle with the same properties as the particle, such as mass, but with opposite values of the additive quantum numbers. The fermions consist of *quarks* and *leptons*. The fermions are classified according to their interactions.

There are six types of quarks, up (u), down (d), charm (c), strange (s), top (t), bottom (b) and six types of leptons, electron (e), electron neutrino (ν_e), muon (μ), muon neutrino (ν_μ), tau (τ), tau neutrino (ν_τ).

The pairs from each classification are grouped together and form a generation with corresponding particles in order to present a similar physical behavior.

The six quarks are paired in three generations as shown in Figure 1.3.



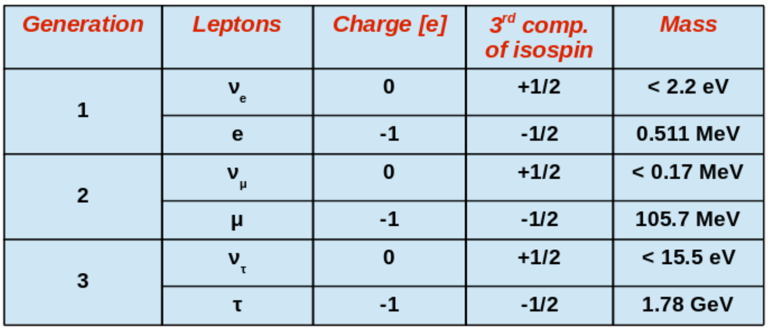
Generation	Quarks	Charge [e]	3^{rd} comp. of isospin	Mass
1	u	+2/3	+1/2	2.3 MeV
	d	-1/3	-1/2	4.8 MeV
2	c	+2/3	+1/2	1.275 GeV
	s	-1/3	-1/2	95 MeV
3	t	+2/3	+1/2	173. 34 GeV
	b	-1/3	-1/2	4.18 GeV

Figure 1.3: An overview of the properties of six quark particles in the SM [16].

The first generation includes light and stable particles, subsequent generations include unstable particles. Each quark can have three possible colours: red, green, blue.

Correspondingly, the six leptons are paired in three generations as shown in Figure 1.4. The leptons, unlike the quarks, do not have any colour charge.

There are four main interactions: gravity, the electromagnetic interaction, the weak interaction and the strong interaction.

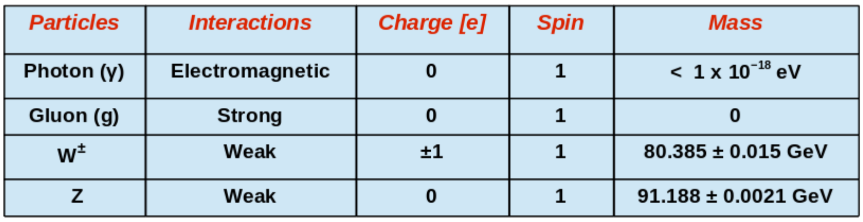


Generation	Leptons	Charge [e]	3^{rd} comp. of isospin	Mass
1	ν_e	0	+1/2	< 2.2 eV
	e	-1	-1/2	0.511 MeV
2	ν_μ	0	+1/2	< 0.17 MeV
	μ	-1	-1/2	105.7 MeV
3	ν_τ	0	+1/2	< 15.5 eV
	τ	-1	-1/2	1.78 GeV

Figure 1.4: An overview of the properties of six lepton particles in the SM [16].

- *Gravitational interaction*: a natural interaction by which all objects with mass gravitate to each other, for example stars, planets, galaxies and even subatomic particles. This interaction is much weaker than the electromagnetic, strong, and weak interactions when applied to particles and therefore is not considered within the SM.
- *Electromagnetic interaction*: occurs between electrically charged particles. Photons act as the electromagnetic force mediator.
- *Strong interaction*: many times stronger than electromagnetic interactions and much stronger than weak interactions. It binds quarks together to protons, neutrons and other *hadron* particles. A *hadron* is a composite particle made of quarks that are held together by the strong force. Gluons act as the strong force mediator.
- *Weak interaction*: takes care of the radioactive decay of unstable nuclei and for interactions of neutrinos and other leptons with matter. The W, Z bosons act as weak force mediators. The W bosons have a positive or negative electric charge (W^+: +1 and W^-: -1) and are antiparticles of each other, while the Z boson is electrically neutral and is its own antiparticle. All three particles have a spin of 1.

All described above interactions are shown in Figure 1.5.



Particles	Interactions	Charge [e]	Spin	Mass
Photon (γ)	Electromagnetic	0	1	< 1×10^{-18} eV
Gluon (g)	Strong	0	1	0
$W^{\pm}$	Weak	±1	1	80.385 ± 0.015 GeV
Z	Weak	0	1	91.188 ± 0.0021 GeV

Figure 1.5: An overview of different particles and its interactions in the SM [16].

1.4 Unification

The primary goal of Particle Physics is to bring together or unify the fundamental particles and their interactions.

In the 1860s, Maxwell showed that electricity and magnetism together constitute one and the same phenomenon called electromagnetism.

In the 1970s, Glashow, Salam and Weinberg managed to unify the Weak and Electromagnetic interactions into one called the Electroweak force. The Nobel Prize in physics was later awarded to them in 1979 for their "contribution to the theory of the unified weak and electromagnetic interaction between elementary particles, including, inter alia, the prediction of the weak neutral current" [17].

1.5 The Higgs Mechanism

In the SM model, the Higgs mechanism [18] plays an important role as it explains the generation mechanism of the mass property for gauge bosons and fermions.

In 1964, the Higgs mechanism was proposed by three independent groups of scientists (Robert Brout and François Englert, Peter Higgs and Gerald Guralnik, C. Richard. Hagen, and Tom Kibble), who proposed different but related approaches about how mass could arise in local gauge theories.

In 2010, all six physicists were awarded the J. J. Sakurai Prize for Theoretical Particle Physics. July 4 2012 was announced that the new particle observed at CERN in the mass region around 126 GeV. The observed particle corresponds to the previously predicted the Higgs boson in the SM. The 2013 Nobel Prize in Physics was awarded to Peter Higgs and François Englert.

Without this mechanism, all fundamental bosons would be massless, while, according to experimental results it is clear that the mass of the W and Z bosons are 80 and 91 GeV, respectively.

In the SM model, the Higgs mechanism refers to a generation of masses for $W^{\pm}$ and Z bosons through electroweak symmetry breaking. It also generates masses for all fundamental fermions.

A solution for describing the mechanism is to add a quantum field (the Higgs field) to the SM. In the conditions of high temperature, the added field causes spontaneous symmetry breaking during interactions, which causes bosons to have a mass.

The determination of Higgs potential can be represented by the following Lagrangian 1.2:

$$L = 1/2\partial_{\mu}\phi\partial^{\mu}\phi - V(\phi), V(\phi) = 1/2\mu^2\phi^2 + 1/4\lambda\phi^4 \tag{1.2}$$

The graphical representation of the Higgs potential is shown in Figure 1.6.

The shape of the Higgs potential depends on the sign of μ^2 as λ is a scalar and is always positive $\lambda > 0$. If the $\mu^2 < 0$, then the minimum of the potential is 0 ($|\phi_0| = 0$). If the $\mu^2 > 0$, the $V(\phi)$ has a minimum when $\partial V/\partial\phi = \mu^2\phi + \lambda\phi^3 = 0$ as it is shown in the Figure 1.6 a) b) correspondingly.

$|\phi_0|^2 = \mu^2/\lambda = \nu^2$, where ν is the vacuum expectation value *(vev)* of the scalar field ϕ.

As all the other fundamental fields, the Higgs field also has an associated particle, called the *Higgs boson*. In the SM, the Higgs boson is a particle that does not have any spin and electric charge. It is also decaying into the other particles easily.

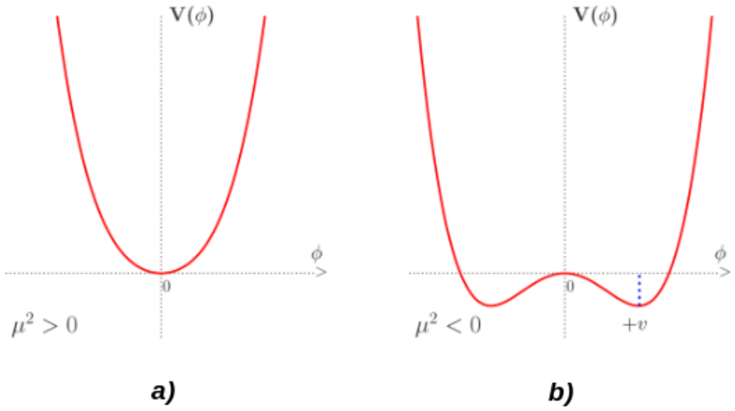
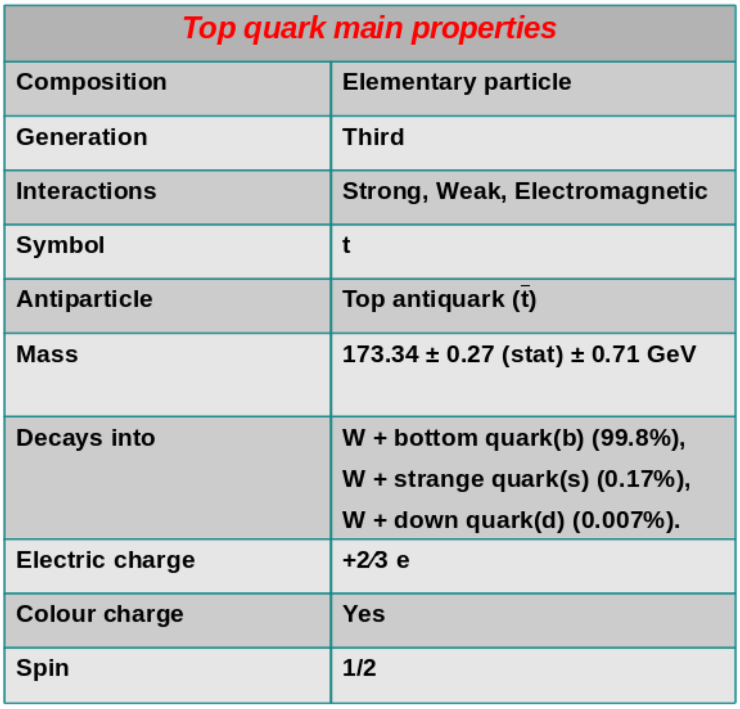


Figure 1.6: The graphical representation of the Higgs potential $V(\phi) = 1/2\mu^2\phi^2 + 1/4\lambda\phi^4$ for the real field ϕ. The a) is the case where $\mu^2 > 0$. The b) is the case where $\mu^2 < 0$.

1.6 The Top Quark

The top quark (t) is the heaviest particle among all other observed fundamental particles. The top quark is a fundamental fermion with spin-1/2 and an electric charge of +2/3. The mass of top quark is $173.34 \pm 0.27(stat) \pm 0.71 GeV(syst.)$ [19]. Due to its large mass, it is the quickest particle to decay and the lifetime of the top quark is very small $\approx 0.5 \times 10^{-24}$ s. It decays before it can hadronize. The Figure 1.7 represents a general overview about the top quark properties.



Top quark main properties	
Composition	Elementary particle
Generation	Third
Interactions	Strong, Weak, Electromagnetic
Symbol	t
Antiparticle	Top antiquark ($\bar{t}$)
Mass	173.34 ± 0.27 (stat) ± 0.71 GeV
Decays into	W + bottom quark(b) (99.8%), W + strange quark(s) (0.17%), W + down quark(d) (0.007%).
Electric charge	+2⁄3 e
Colour charge	Yes
Spin	1/2

Figure 1.7: Main properties of top quark.

The top quark has its aniparticle called the antiquark or antitop $(\bar{t})$.

The top quark interacts through the strong interaction. However the decay of the top quark is only possible via the weak interaction. It can decay in three possible ways:

- *W boson + bottom (b) quark*: the most frequent case ($\approx 99.8\%$),
- *W boson + strange (s) quark*: the rare case ($\approx 0.18\%$),
- *W boson + down (d) quark*: the most rare case ($\approx 0.02\%$).

The top quark can be produced either in pairs (pp->$t\bar{t}$) or as single quarks. The $t\bar{t}$ pairs are produced much more often than single quarks.

QCD [9] describes all $t\bar{t}$ productions. When two protons collide, they break into quarks and gluons. The $t\bar{t}$ pairs are the result of collisions between the quarks and gluons.

At the LHC, the $t\bar{t}$ production is dominated by gluon-gluon (gg) fusion ($\approx 80\%$ of cases, at $\sqrt{s} = 7$ TeV), while at the Tevatron, it is dominated by quark-antiquark ($q\bar{q}$) annihilation ($\approx 90\%$ of cases, at $\sqrt{s} = 2$ TeV). The Figure 1.8 displays, the leading order (LO) Feynman diagrams of gg fusion and $q\bar{q}$ annihilation.

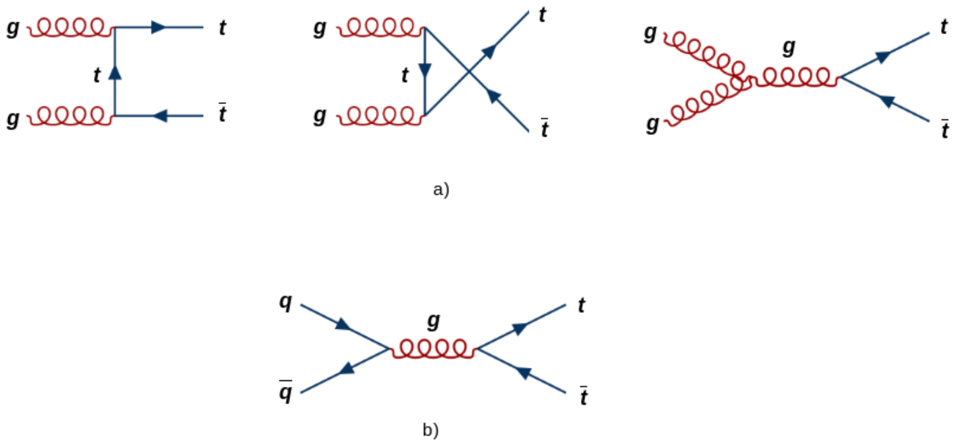


Figure 1.8: Leading order Feynman diagrams for $t\bar{t}$ production gg fusion (a) and $q\bar{q}$ (b) annihilation.

The production of single top quarks occurs through the weak interaction. It can be produced in three possible ways called channels: *t-channel*, *Wt-channel* and *s-channel*. The t-channel is the most observed type of channel at the LHC.

These channels are displayed in Figure 1.9.

Because of its large mass, the top quark can be the key for understanding the origin of generated particle masses by the mechanism of electroweak symmetry breaking.

1.7 Beyond the Standard Model

In Particle Physics, the SM model is the most successful theory, but is not a perfect one. In spite of all the explanations given by the SM model, there are a number of different phenomena or observational facts that are unexplained. The most interesting once are the following:

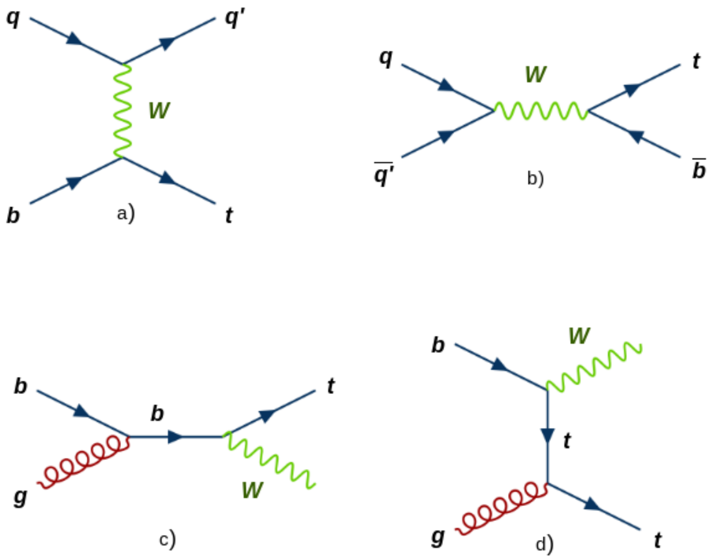


Figure 1.9: Feynman diagrams for single top production in *t-channel* (a), *s-channel* (b), *Wt-channel* (c and d). The q and q' are *up-* and *down*-type quarks.

- *Gravity*: the SM does not explain gravity. Even the most successful theories of gravity are considered incompatible with the SM.
- *Dark matter and dark energy*: the invisible part of the universe is called dark matter and dark energy. According to the cosmological observation, the SM model explains only 5% of the present universe energy. The rest rely on the dark matter and dark energy.
- *Neutrino masses*: according to the SM, neutrinos have no mass. However, neutrino oscillation experiments [10] showed the exact opposite.

Explanation of the phenomenon can modify the human understanding of the world and the SM itself.

1.8 ATLAS

The ATLAS (A Toroidal LHC ApparatuS) [20] experiment is designed to take advantage of the unprecedented energy available at the LHC and observe phenomena that involve highly massive particles which were not observable using earlier lower-energy accelerators. The ATLAS detector (Figure 1.10) is 46 metres long, 25 metres in diameter, and weighs about 7,000 tonnes; it contains some 3000 km of cable. The experiment is a collaboration involving roughly 3,000 physicists from over 175 institutions in 38 countries.

The ATLAS detector has been designed so that it can cover a wide range of issues. It allows accurate measurements within the parameters of the SM, as well as the discovery of a new physics phenomena that may occur at higher energies. The search of Higgs boson is an example for determining the performance of various ATLAS detector subsystems.

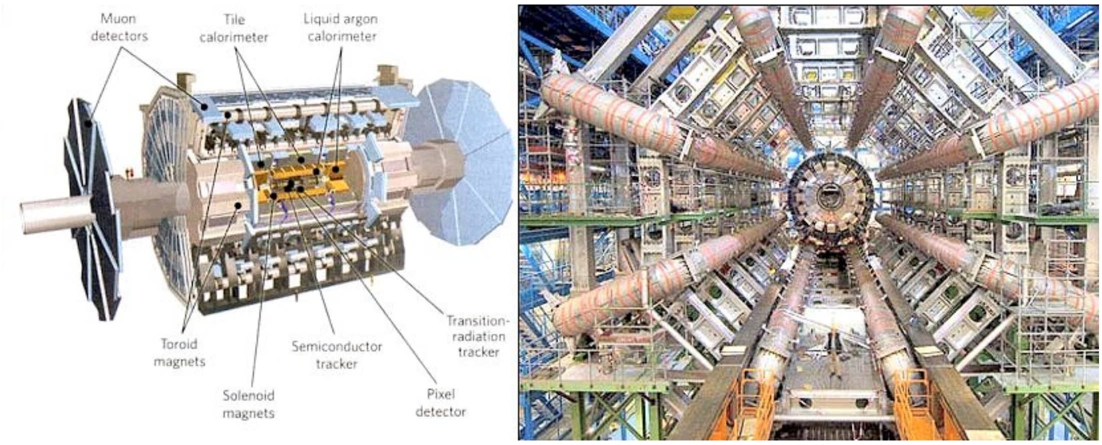


Figure 1.10: ATLAS detector.

The ATLAS detector subsystems are located around the point of interaction and have a similar structure as the layers of an onion.

Beams of particles collide with each other forming a large number of subatomic particles which are spread in all possible directions of the detector. Different sub-detectors are designed to identify the types of particles, as well as to record the momentum, path and energy of the particles (Figure 1.11). The only particle that does not leave any trace in the detector is the neutrino.

Measuring the energy of different particles and its properties is the main task of the ATLAS detector and its entire infrastructure.

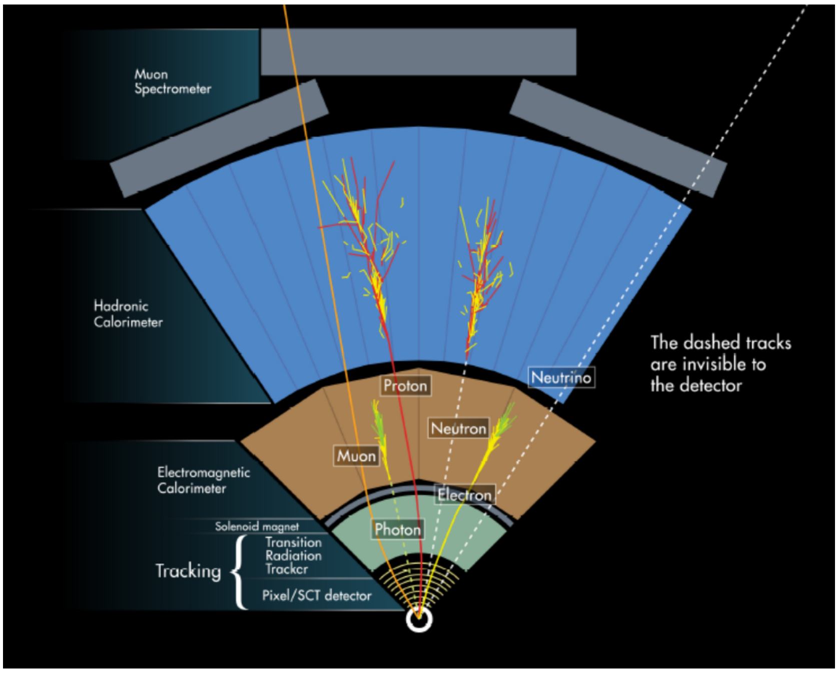


Figure 1.11: ATLAS detector and its subsystems.

Detectors generate unimaginably large amounts of data. To cope with this fact, ATLAS uses an advanced "trigger" system [22] that filters the events according to their importance, before they can be recorded. Even though the amount of data, that needs to be stored, is huge.

1.9 Worldwide LHC Computing Grid (WLCG)

The Worldwide LHC Computing Grid (WLCG) [13] project is a global collaboration of more than 170 computing centers in 41 countries, connected by grid infrastructures. WLCG was built on a grid concept, which was previously proposed by Ian Foster and Carl Kesselman in 1999 [24].

The goal of the WLCG is to provide global computing resources in order to distribute, store and analyse approximately 15 petabytes [25] of data generated by the LHC experiments at CERN every year (Figure 1.12).

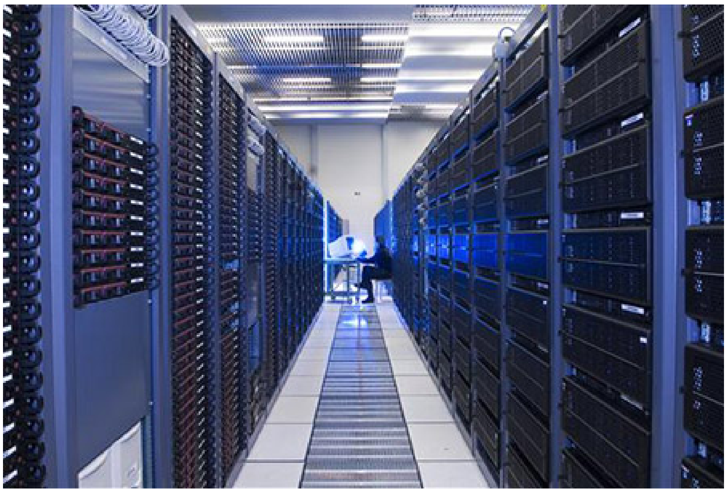


Figure 1.12: CERN grid computing centers.

A number of scientists from all over the world from four different LHC experiments (ATLAS, CMS, ALICE, LHCb) are actively accessing and analysing data. The computing system that is designed to handle the data has to be very flexible. WLCG provides access to the computing resources which include compute clusters, storage systems, data sources and necessary software tools.

Scientists make a script or a so-called "job", submit it to the WLCG and wait until it executes and returns the result. The jobs that scientists create can be very different, for example file transfers, different and complex calculations, etc. The computing grid establishes the identity of the users, checks their credentials and if the user is a member of a collaboration, then he/she is allowed to run their job. Users do not have to worry about the location of computing resources – they can tap into the grid's computing power and access storage on demand.

1.9.1 Tier Structure

Dealing with tens of petabytes of data is not an easy task. It requires careful planning and organisation. Different computing centers have different resources and geographical locations. The tiered structure [26] allows to group these computing centres according to their location for serving a community of more than 8000 scientists. The role of computing centers are very important. They basically store all necessary data (raw and analysed) from all over the world.

The LHC computing center has a four layer tiered structure (Figure 1.13).

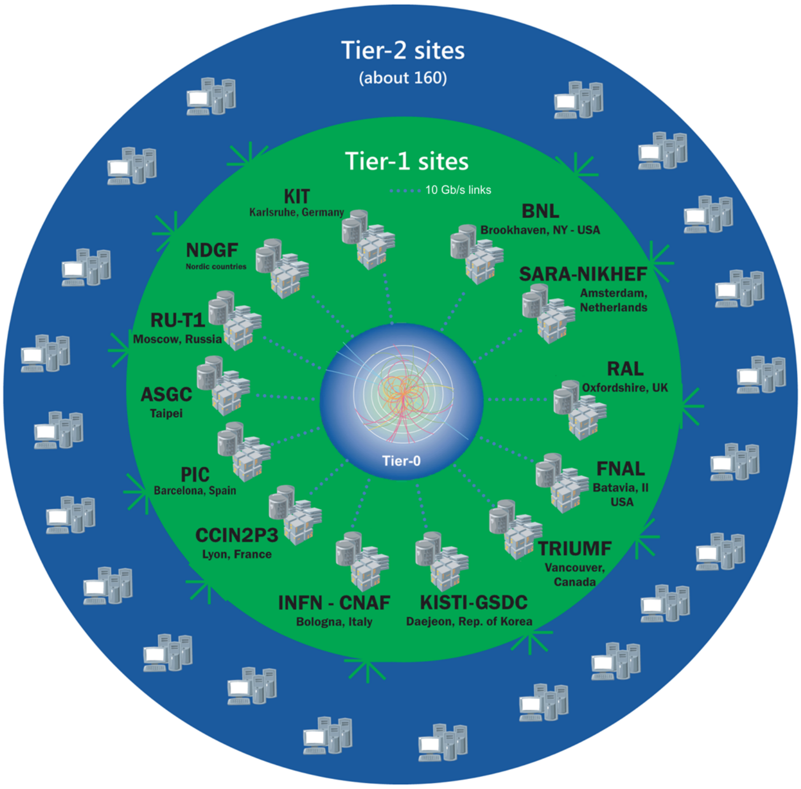


Figure 1.13: CERN tier structure.

The *Tier-0* centers are the main computing centers located at CERN in Geneva, Switzerland and at the Wigner Research Centre for Physics in Budapest, Hungary. The Tier-0 center in Budapest is designed as an extension of the CERN Tier-0 center. These two computing centers are connected with each other by 100 Gbit/s data links.

Here, the raw or original data are stored, which are coming directly from different experiments. It is required that all raw or original data are stored in the permanent storage. Then, initial processing of the data is performed on the site to provide rapid feedback to the operation. Afterwards, the data are sent to the other computing centers for further analysis. The role of the computing centers is to provide analysis capacity for the scientists/users. Some resources are usually intended for the local users, others which are intended for the simulations can be provided for the experiments.

The *Tier-1* computing centers are 13 in total. These are very large national computing centers. They receive data directly from CERN and provide additional permanent storage. The computing centers also provide computing resources for the data reprocessing. Having a special role they provide reliable services to the community such as databases and catalogs.

The *Tier-2* centers are 160 in total. Their role is to provide the storage capacity and computing resources for specific analysis tasks. Typically they are associated with a large disk storage to provide temporary storage for data that is required for analysis.

The *Tier-3* centers are the smallest computing centers located at different universities and laboratories. Their main role is to provide local clusters or individual PCs.

1.9.2 Data Flow

The main source for the computing model is the Event Filter (EF) [27]. The input and output data for the EF requires different network connection speed. For example, for the input data, approximately a 10x10 Gbps network connection speed with very high reliability and for the output data, approximately a 320 MB/s (3 Gbps) connection to the first-pass processing facility is required. For remote sites the network connection speed requirement is higher, approximately 10 Gbps to the remote site.

The streaming of data at the EF should be reserved and for this reason the computing model purposes usage of a single stream, which contains all physics events passing from the Event Filter to the Tier-0. There are also other supportive streams, for example calibration, express and pathological streams.

The *calibration stream* contains calibration trigger events and is used to produce sufficient quality calibrations in order to allow a first-pass processing with the minimum latency.

The *express stream* contains approximately 5% of the full data. This stream is used for the improvement of the data reconstruction speed. It is designed to provide early access to the raw data and the calibration streams.

The *pathological stream* contains pathological events such as failures, for example, events that fail in the EF. Typically they pass standard Tier-0 processing but in case of failure they get special treatment from the development team.

After raw data arrives at the input-disk buffer of the Tier-0 site, they pass several steps, such as:

- data copied to the CERN Advanced STORage manager (CASTOR) tape disk [28] at CERN,
- data copied to one of the Tier-1s permanent mass storage,
- corresponding calibration stream events takes care of calibration and alignment of the raw data,
- after having an appropriate calibration, first-pass reconstruction starts to run on the primary event stream. This stream contains all physics trigger information and the result is stored in CASTOR tape disk,
- copy data to each Tier-1 site,
- replicate the copied data to each Tier-2 site.

The raw data transfer to the Tier-1 sites is a very important aspect. These sites are the main sources for the later data reprocessing and data reconstruction. These sites can also be used as an additional capacity, in case there is a backlog of first-pass processing at the Tier-0.

1.9.3 Analysis of High Energy Physics Data

In HEP experiments the analysis of data requires a complicated chain of data processing and data reduction. The huge amounts of data recorded by experiments have to be reconstructed and processed before it will be available for scientists from all over the world. Data needs to pass several stages.

The first stage is the so-called *raw data*. This data are generated by the detectors of the experiment. Raw data consists of the raw measurements of the detectors. These can be time measurements, channel numbers, charge depositions and any other signals. Integrated over the time, data taking during a year is expected to be a petabyte. The data has to be stored in a safe way on a permanent storage.

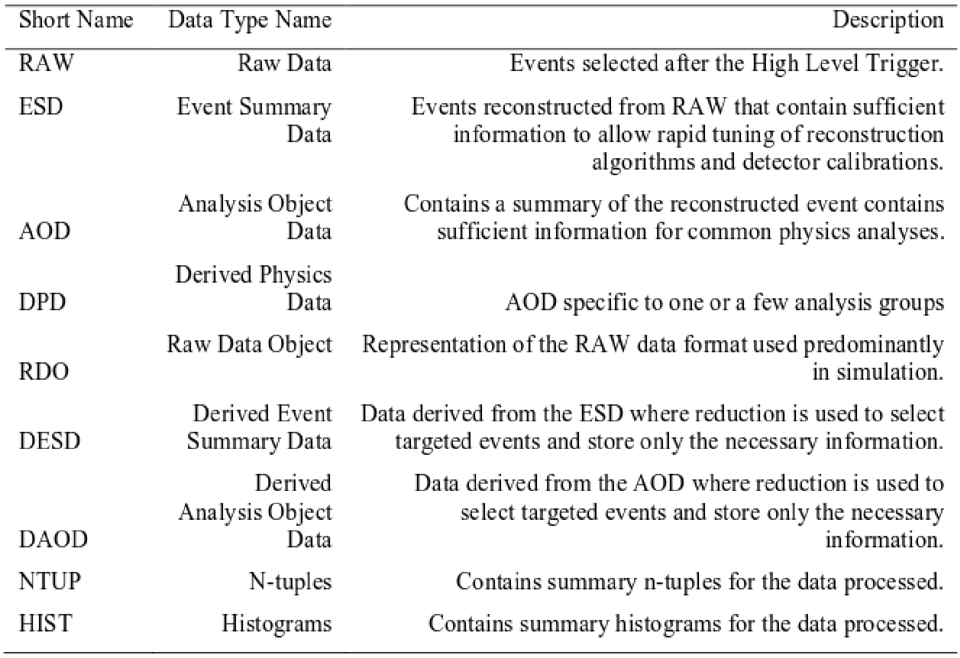
The second stage is called *reconstruction*. During this stage the data are processed. Raw numbers are converted into physical parameters. Later, pattern recognition systems translate the parameters of observed particles to their moments or energy. The output of this processing step is called *reconstructed data*.

The reconstructed data then has to be distributed to the scientist for further detector studies and specific analysis.

To simplify the analysis procedure only the most valuable dimensions are stored in separate streams. These data are called AOD (Analysis Object Data). The size of the events is significantly reduced. It is expected that the amount of AOD per year is approximately 100 TB.

In the process of the analysis of the data, scientists perform further data reductions. The scientists define their own ad hoc data format. In ATLAS these datasets are known as DPD (Derived Physics Data), in CMS as PAT Skims (Physics Analysis Toolkit).

The data types used by the ATLAS experiment is shown in Figure 1.14.



Short Name	Data Type Name	Description
RAW	Raw Data	Events selected after the High Level Trigger.
ESD	Event Summary Data	Events reconstructed from RAW that contain sufficient information to allow rapid tuning of reconstruction algorithms and detector calibrations.
AOD	Analysis Object Data	Contains a summary of the reconstructed event contains sufficient information for common physics analyses.
DPD	Derived Physics Data	AOD specific to one or a few analysis groups
RDO	Raw Data Object	Representation of the RAW data format used predominantly in simulation.
DESD	Derived Event Summary Data	Data derived from the ESD where reduction is used to select targeted events and store only the necessary information.
DAOD	Derived Analysis Object Data	Data derived from the AOD where reduction is used to select targeted events and store only the necessary information.
NTUP	N-tuples	Contains summary n-tuples for the data processed.
HIST	Histograms	Contains summary histograms for the data processed.

Figure 1.14: The data types used by the ATLAS experiment. Taken from [23].

Monte Carlo (MC) events gets special treatment. In the process of the analysis it is required to carefully study the sensitivity and coverage of the detectors. The basis of such studies are thorough and detailed simulation of events.

1.10 Phase-2 Upgrade

The LHC is the largest and most powerful particle accelerator in the world (see Section 1.2). In the period between 2013 and 2015, the LHC was in a technical stop. This two-year period was enough to prepare the machine for the running at almost the double of the energy of LHC's Run 1. The machine has been restarted and is running with proton-proton collisions at a center of mass energy of 13 TeV (Run 2 phase).

The preparation phase for the LHC Run 2 [2] was not only the improvement of the machine itself, but also is a long and complex preparation for all the necessary hardware and software including increment of the storage capacity, changes an existing software versions, maintenance, monitoring, etc. During the shutdown the necessary systems (hardware/software) were verified and some were renovated and upgraded.

The main improvements for different LHC experiments are described below.

ALICE

In the ALICE experiment, which studies the quark-gluon plasma, its 19 sub-detectors have been improved. Among these was the electromagnetic calorimeter, used for measuring the energy of electrons, positrons and photons produced by the collisions. The range of detection was extended.

Several modules were added in other sub-detectors. Even the cable of tens of kilometres were replaced as a part of complete reparation of the electrical infrastructure.

The trigger and data-acquisition systems were improved and now the data registration has increased by almost a factor of two. The storage capacity has increased by almost a factor of two as well.

ATLAS

The ATLAS detector has improved as a fourth layer of pixels in its pixel tracker was added. General improvements were also done in the muon detectors and calorimeters. Here the basic infrastructure was tuned (including the electrical power supply and the cooling systems).

The trigger and data-acquisition systems were improved as well. ATLAS is now able to log approximately twice the data than in LHC Run 1. In addition, the simulation, reconstruction and data-analysis software which are mainly used by different physicists to run their own analysis was also upgraded and renewed.

CMS

For the CMS experiment, the major work was done in its tracker. Now it can operate in lower temperature. It was also equipped with a new leak-tightness system and renovated the cooling system.

Other improvements were also applied. The beam tube, where collisions occur, was replaced by a smaller diameter tube. A new sub-detector, the pixel luminosity telescope was mounted on both sides of the detector which is increasing the experiment's ability to measure luminosity. New muon chambers were installed and the hadron calorimeter was fitted to the upgraded photodetectors. The last point is that the trigger and data-acquisition systems were improved and the corresponding software and the computing system were significantly changed in order to reduce the needed time for producing the analysis datasets.

LHCb

HeRSChel is the forward shower counter project for LHCb. The LHCb experiment which investigates beauty particles, added a HeRSChel detector along the beam line for identifying rare processes in which particles are observed inside the detector but not along the beam line itself. The beam pipe was replaced with a new one.

Chapter 2

Grid Computing

2.1 The Concept of Grid Computing

Grid computing [29] was developed in the early 1990s for making computing power easy to access. Grid computing is a collection of computers located in various places to solve complex tasks. The grid can be also seen as a distributed system.

In comparison to other systems, the grid is focused on the performed amount of work over a period of time, while the others are mostly focused on the greater performance in terms of processing floating-point operations per second (FLOPS) in the system. Floating-point operations are usually calculated by certain algorithms or computer programs. FLOPS is a measure of computer performance and is important for scientific calculations.

The grid is an evolutionary technology that uses existing IT infrastructure to provide high-throughput computing.

One of the most important terms of a grid system is "virtualisation". This refers to the direct integration of geographically distributed systems. This is very important as users do not need to know where the computer is physically located. This also means that there is a single access point for users to enter the grid system and to submit their requests. Users just need to submit the request from the grid user interface (UI) (see Section 2.1.3) and then it is up to the grid system to allocate the available computing resources for the dedicated requests.

The grid infrastructure introduces the concept of virtual organisations (VO) [31]. A VO is a collection of individuals or institutions that are sharing the common grid computing resources (Figure 2.1).

Based on the concept of virtual organisations, we consider three conditions that provide the basis for the understanding of the grid system. The first condition is virtualisation, which was explained above.

The second condition is heterogeneity. Talking about VOs indicates a multi-institutional entity. These organisations can have different resources in terms of hardware, network and operating systems. So it should be clearly defined that a VO is a collection of heterogeneous resources.

The third condition is flexibility. Different organisations can leave or join a VO according to their requirements and convenience. So a VO is a dynamic entity.

These three terms explain why grids have specific requirements in contrast to other distributed systems. Grids enable collaboration between multiple VOs in terms of resource sharing. This collaboration is not only focused on the exchange of files between each other but also provides direct access to the computing resources. Each of these VOs can have different policies and administrative control. All VOs are part of a big grid system. Resources that are shared between VOs may be data, special hardware,

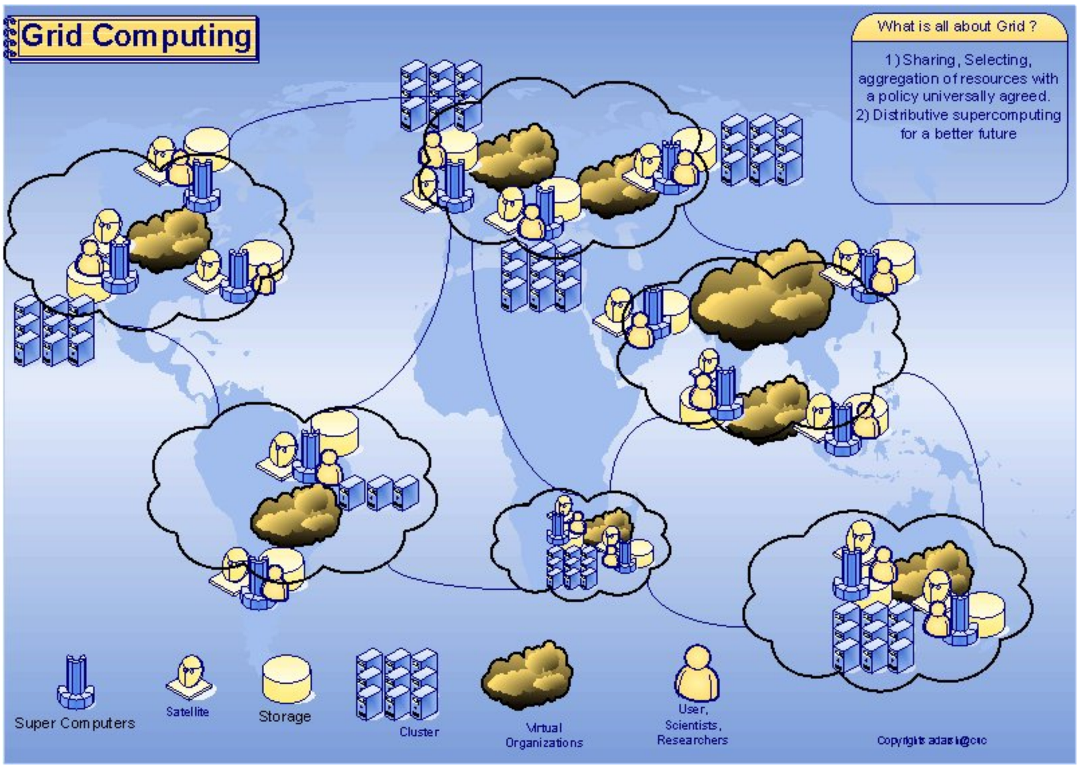

Grid Computing
What is all about Grid ?
1) Sharing, Selecting, aggregation of resources with a policy universally agreed.
2) Distributive supercomputing for a better future
Super Computers
Satellite
Storage
Cluster
Virtual Organizations
User, Scientists, Researchers
Copyright: adarsh@cic

Figure 2.1: Different VOs in the grid. Taken from [30].

analysed and distributed data in the grid. The members of the grid can be part of multiple VOs at the same time (Figure 2.2).

Grids can be used to define a security policy for the members with the ability to prioritise resources for different users.

Ian Foster describes a three point checklist [32] to describe a grid. According to it, a grid should provide resource coordination without centralised control, it should be based on open standards, and it should provide a nontrivial quality of service. A grid can be used for computational purposes (computational grid), for storage of data on a large scale (data grid), or a combination of both.

2.1.1 Grid Authentication/Authorisation

As we discussed, VOs are a group of geographically distributed individuals or institutions that are sharing their common resources between each other. Resource sharing is controlled by a set of rules or policies that define the condition and scale of that sharing. The flexibility of VOs brings the problem of security. Typically, the well-known terms for computer security are authentication, authorisation and confidentiality.

Grid infrastructure provides security technologies using VO as a bridge among the grid users and grid resources. One part of Globus Toolkit [33] is Grid Security Infrastructure (GSI) (see Section 2.1.1.5) that implements grid security functionality.

The Globus Toolkit is a primary technology for the grid and allows users to securely share the same computing resources and other necessary tools. The toolkit includes security, software library services for the resource management and monitoring.

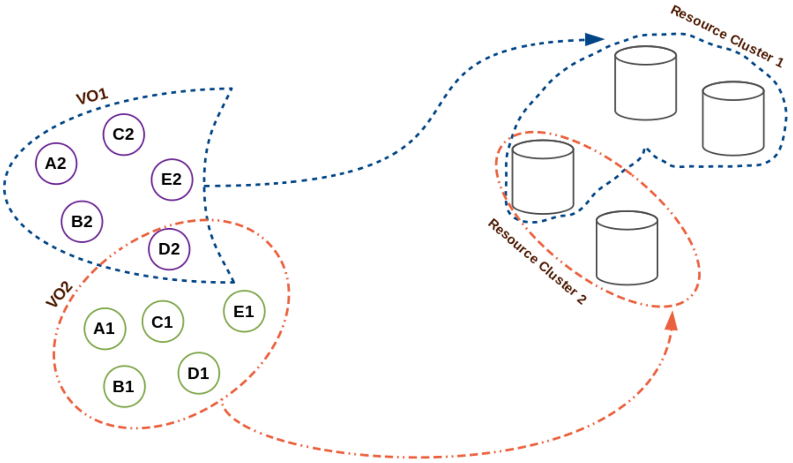


Figure 2.2: Different VO members can share the same grid resource.

2.1.1.1 Authentication

Authentication establishes an identity of an object in the network. The object can be a process, a resource or a user. For example if you need to enter another computer or website, which requires a particular identity, then you need to identify yourself. This process is called authentication.

The simplest type of authentication for websites or computers can be a *username and a password* authentication [34]. Typically, a website authentication procedure is the following: A website should have a database of usernames, passwords, user roles configured on the web or application server. After this, the website can check authenticated users and grant access. However, *username password* based authentication is not very secure as a user sends the data as plain text. If someone intercepts the transmission, the user name and password can be easily decrypted.

More complicated and more secure type of authentication is *Kerberos* [35], which is used for client/server applications by using symmetric key cryptography.

The technology that plays the key role in the authentication of grid systems is *Public Key Infrastructure (PKI)* [36]. PKI describes the security system used to identify objects through the use of X.509 certificates [37]. X.509 certificates are issued by highly trusted organisations known as Certifying Authorities (CAs) [38]. Different VOs have agreed on the use of different CAs. Hundreds of users can use the same grid system at the same time using their own user credentials. More detailed information is provided in the following sections.

2.1.1.2 Public Key Infrastructure (PKI)

The Public Key Infrastructure (PKI) [36], [39] is built to provide secure communication in public networks or the internet. It uses a public/private key pair structure. For each user the message encryption and decryption should satisfy the equalities shown in Equations 2.1 and 2.2.

$$D(E(Msg)) = Msg, \tag{2.1}$$

$$E(D(Msg)) = Msg, \tag{2.2}$$

where *Msg* stands for any message or text, E is the encryption key and D is the decryption key.

When the user A wants to send a *signed document* to the user B, the following steps need to be followed:

- Transform the message into a signed message (*MsgSigned*) by using his/her decryption key, such as $MsgSigned = D_A(Msg)$,
- Send to the user a signed ciphertext. User A uses the encryption key of the user B, which can be found in the public file, to generate a signed ciphertext ($CSigned = E_B(MsgSigned) = E_B(D_A(Msg))$).

The actions of the user B are the following:

- The user B reduces the signed ciphertext (*CSigned*) to a signed message (*MsgSigned*) with his/her decryption key, such as

  $D_B(CSigned) = D_B(E_B(MsgSigned)) = MsgSigned.$

- The user B decrypts the signed message by using the encryption key of the user A, which can be found in the public file, such as $E_A(MsgSigned) = Msg$.

The schematic view of the above mentioned example is shown in Figure 2.3.

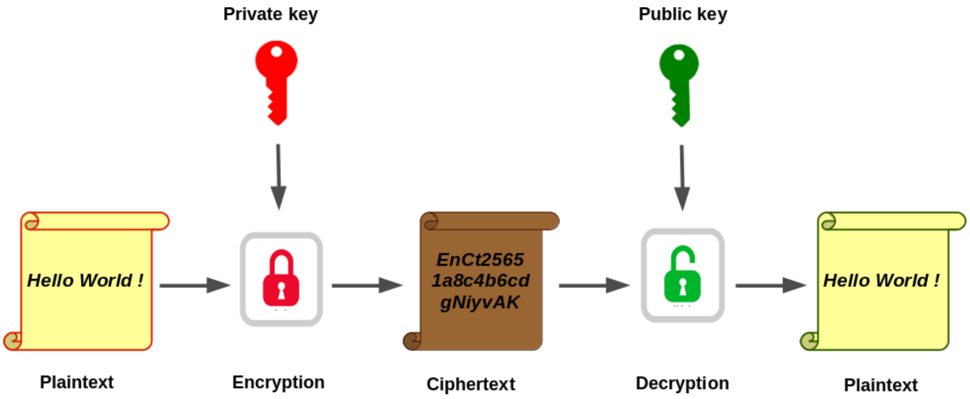


Figure 2.3: The schematic view of the example. The user A sends a *signed document* to the user B.

The CA issues a digital certificate to users (individuals/organisations). This certificate uniquely identifies a user. The digital certificate follows the structure specified by the X.509 system. The name of the user in the X.509 system is unique and it is related to its public key by a CA. The certificate private key is kept with the owner of the certificate and the public key is available for public usage. The data signed by the private key can be decrypted only using the public key and vice-verse.

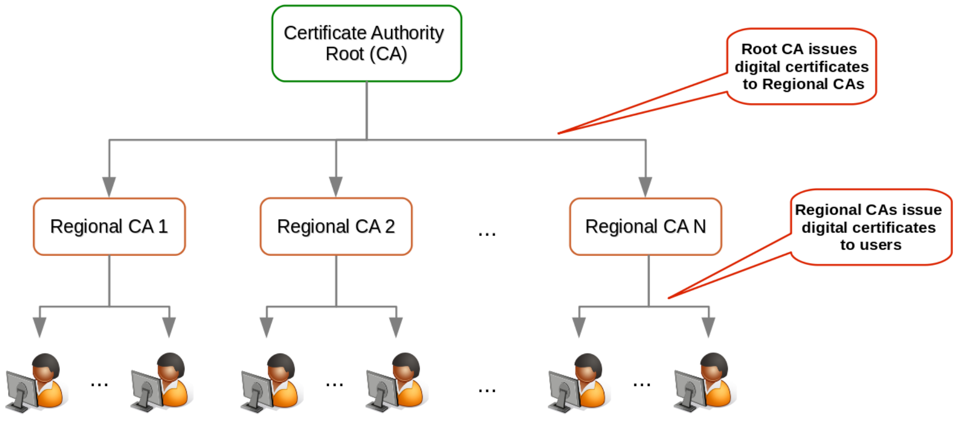

Certificate Authority Root (CA)
Root CA issues digital certificates to Regional CAs
Regional CA 1
Regional CA 2
...
Regional CA N
Regional CAs issue digital certificates to users
...
...
...
...

Figure 2.4: Hierarchy structure for PKI.

The hierarchical structure of the Public Key Infrastructure is shown in Figure 2.4.

At the lowest level of the tree are the end users who are issued with the digital certificate.

In the middle level of the tree are the CAs who are authorised to issue certificates on a regional level. Regional CAs can be small or big organisations, institutions and countries.

At the top level of the tree are the biggest CAs or so-called Root Certificate Authority (Root CA) [40] that are able to provide certificates to smaller CAs. Root CAs are trusted by everyone. They are specially meant for that. The Root CA is identified by the root certificate. The root certificate is a self-signed or unsigned public key certificate. The private key of the root certificate is used to sign all other certificates. All certificates below the root certificate inherit directly the trustworthiness of the root certificate.

The self-signed certificate is a certificate of identity signed by the same person whose identity is certified. From a technical point of view, the self-signed certificate is signed by a user's own private key. Users have to wait until the certificate is signed or approved by trusted CAs.

The CAs in the PKI are also responsible for publishing the Certificate Revocation List (CRL). This list contains the serial number of invalid or expired certificates of the users.

2.1.1.3 X.509 End User Certificates

The Globus Toolkit middleware is a dominant middleware for grid systems.

The Grid Security Infrastructure (GSI) (see Section 2.1.1.5) is one part of the Globus Toolkit that provides the fundamental security services for grid systems. The GSI provides libraries and tools for authentication and message protection that use standard X.509 public key certificates, public key infrastructure (PKI), the SSL/TLS protocol [41] and X.509 Proxy Certificates.

X.509 certificates [37] are used to identify users in the grid. In the grid, each user is identified by a unique X.509 certificate and for any grid activities he/she needs to use the same certificate. Each certificate contains the public key of a user and is signed with the private key of a trusted CA. X.509 defines alternative authentication protocols based on the use of public-key certificates. The user certificates are assumed to be created by some trusted CA and placed in the directory by the CA or by the user. The directory itself is not responsible for the creation of public keys or for the certification function; it only provides an easily accessible location for users to obtain certificates.

User certificates generated by a CA have two main characteristics:

- Any user that knows the public key of the CA can verify the encrypted public key of the user.
- Besides the CA, no one can modify the certificate without being detected.

The format of the X.509 certificate is described below (Figure 2.5).

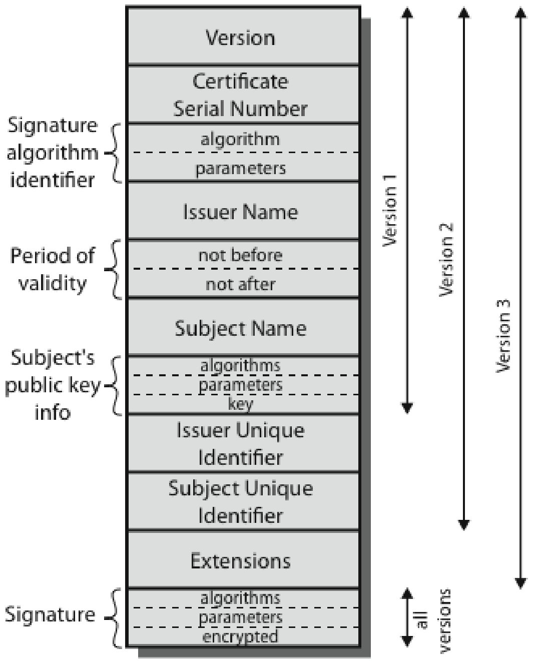


Figure 2.5: X.509 end user certificate format. Taken from [42].

- *Version*: specifies the version of the X.509 certificate.
- *Certificate Serial Number*: CA assigns a unique serial number for the certificate. The CA's name and the serial number uniquely identify the certificate.
- *Algorithm*: identifies the algorithm that was used for the certificate. For example RSA can be used for the generation of the public/private key pair and md5 for the hashing algorithm.
- *Issuer Name*: mentions the name of the CA who signed the certificate.
- *Validity*: consists of two fields "not valid before" and "not valid after". Used for specifying the duration of the valid certificate.
- *Subject Name*: contains the name of the owner of the certificate.

- *Subject public key info*: consist of two fields: "Public key algorithm" and "public key". In the "Public key algorithm" field is mentioned the algorithm that was used to generate the public/private key pair. In the "public key" field is mentioned the public key of the certificate holder.
- *Issuer Unique Identifier*: an optional field used for specifying the unique ID of the issuer of the certificate (CA).
- *Subject Unique Identifier*: an optional field used for specifying a unique ID of the owner of the certificate.
- *Extensions*: an optional field used for specifying the extension of the basic fields.
- *Signature*: consists of the hash of the entire certificate signed by the private key of the CA. In order to verify the certificate a user can calculate the hash using the algorithm specified in the certificate, then decrypt the hash by using the CA's public key. If these two hash values are the same, then the certificate is original and can be trusted.

X.509 certificates have a validity period (lifetime). Typically the certificate lifetime is one year and a new certificate is issued in advance before the expiration of the old one. In some cases, a certificate may need to be revoked earlier than the expiration date. In this case, the CA has a list (certificate revocation list (CRL)) of all revoked but not expired certificates issued by that CA and issued to other CAs. This list is located in the corresponding directory, signed by the issuer and includes the issuer's name, the list creation date, the next CRL creation date and the revoked certificate entries (serial number and revocation date) (Figure 2.6).

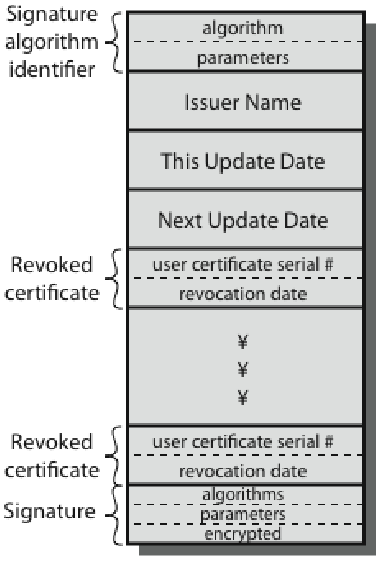


Figure 2.6: X.509 certificate revocation list (CRL) format. Taken from [42].

X.509 certificates are the basis of GSI authentication mechanisms.

2.1.1.4 X.509 Proxy Certificates

Proxy certificates [43] are widely used in security systems. Proxy certificates are based on X.509 public key certificates. The main idea of proxy certificates is to allow single sign-on and delegation of rights for the users.

- *Single sign-on*: the user should be authenticated only once within a domain by using their own X.509 certificate. The creation of the proxy certificate itself is based on the X.509 certificate. Within the domain a proxy certificate can be used multiple times.
- *Delegation of rights*: a user grants partially or all of his/her privileges to another object, in order to perform actions that might have an access control mechanism.

It has a limited validity and the private key can be stored in a simple file system without being encrypted. A proxy certificate has a new public/private key pair. The new certificate contains the owner details and has a specific field where it is mentioned that it is a proxy certificate. A proxy certificate is signed by the owner of the certificate instead of a CA.

If a proxy certificate is used, the authenticating party receives the owner's X.509 certificate and the proxy certificate. The validity of the proxy certificate is checked by the owner's public key. This means to check whether the proxy certificate is signed by the same owner or not. The owner's X.509 certificate has a signature of CA. This may be used to validate the owner's certificate. In this way a chain of trust is established between CA and owner and between owner and proxy certificate.

A proxy certificate is derived as an extension of an X.509 certificate, using the proxy certificate information extension defined in RFC 3820 [44]. A proxy certificate has additional fields which are defined in RFC3820. The extension indicates that the certificate is a proxy and contains the information about any usage restrictions that can be placed by the owner. The structure of the extension is shown in Figure 2.7.

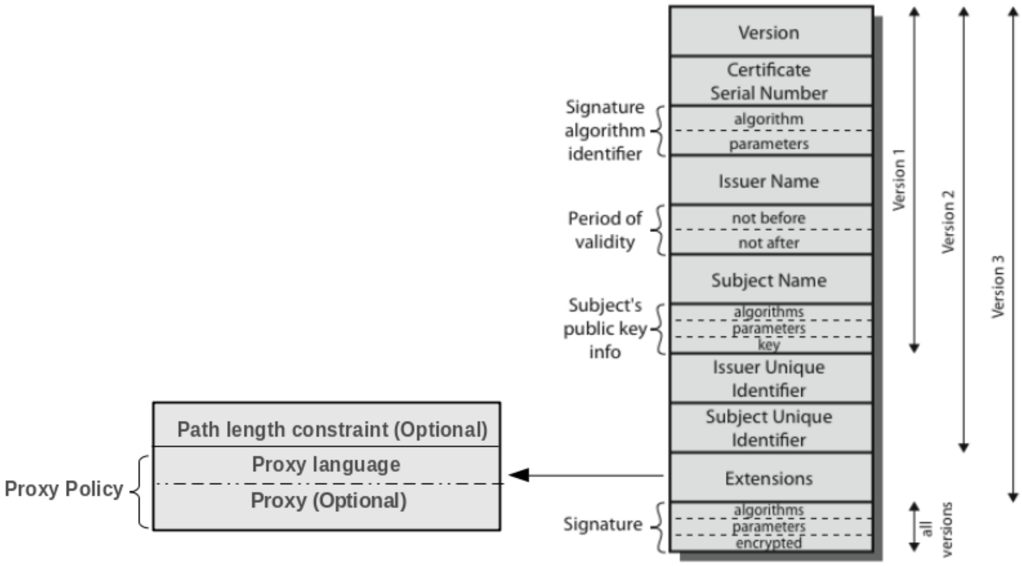


Figure 2.7: X.509 proxy certificate extension [45].

- *Path Length Constraint*: mentions the maximum depth of the proxy certificate path that is allowed to be signed by this certificate.
- *Proxy Policy*: specifies a policy of authorisation within this certificate.
- *Policy Language*: mentions the format of the policy. The language field has two important values: *id-ppl-inheritAll* that points out the usage of unrestricted proxy and *id-ppl-independent* that points out the usage of no rights proxy granted by the owner. In this case the only rights are those that are explicitly granted to it.
- *Policy (Optional)*: mentions other policies.

2.1.1.5 Grid Security Infrastructure (GSI)

Grid Security Infrastructure (GSI) [24] is meant to provide the necessary functionality for the implementation of security in grid systems. GSI is part of the Globus Toolkit. It has been developed to deal with the specific security needs of grid systems, such as: authentication and authorisation.

- *Single Sign-On*: a user should be able to login to the grid system and use the grid resources without requiring any further authentication.
- *Delegation of privileges*: different users can have different privileges in grid systems. For example grid administrators have more privileges than the regular user. Some specific information such as file configurations or software management requires special user privileges.
- *Inter-domain security support*: the grid may contain different security domains geographically located in different places. The grid security must be able to provide support between these domains.
- *Secure Communication*: the grid security should provide secure message exchange within grid users and services.
- *Uniform credentials*: the grid security should have a uniform certificate which will represent credentials of users in the grid. As a uniform certificate GSI uses the X.509 certificate format.

To support these requirements, GSI provides different functions such as, authentication, authorisation, delegation and message protection. Each of these tasks is handled by an open standard.

2.1.1.6 Authorisation

Authorisation is the second step (first step was described in 2.1.1) of trust establishment between two objects in the grid. Authorisation is setting up the privileges for the objects in order to access a dedicated resource. Only after successful authentication, the authorisation can be done.

In grids, the resource owners can have the ability to grant or deny access to a grid system, based on the identity or membership of a VOs. One of the authorisation techniques used in grid systems is the Globus Toolkit Gridmap file [33]. It contains a list of global names of grid users and the corresponding local account names to which they are mapped. The maintaining of the updated version of the gridmap file requires certain effort from the local system administrator. The Community Authorisation Service (CAS) [46] does clear separation between site policies and VO policies. CAS provides a mechanism for a VO to manage multiple independent policies. Each VO consists of a CAS server, which acts as a trusted third party between the users and the resources of the VO.

The Virtual Organisation Membership Service (VOMS) [47] provides an information management authorisation between different VOs. The VOMS provides a database, which contains user roles and capabilities and a set of tools to access and use the database. The VOMS database typically runs on a VOMS server. These tools can create new credentials for the new users. These credentials contain the basic authentication information that can contain the standard grid proxy certificate. Additionally, they also contain the user role information.

2.1.1.7 Confidentiality

Confidentiality means hiding the most valuable or sensitive information from people who should not have rights to access them. Generally grid systems are taking care of valuable or sensitive data, such as experimental, analysed, financial data, etc. This brings up the issue of confidential data protection, which is typically done by cryptography. The provision of data access is discussed within organisations and is managed by the grid administrators. Grid systems supposing usage of remote resources. A remote resource can be either an autonomous machine or a cluster in accordance with the decision of the grid load balancer. Grid systems have specific security requirements, which are different from internet security.

2.1.1.8 Grid Middleware and its Services

In order to use grid resources effectively some middleware is needed. Grid middleware includes APIs, protocols, and software that allows creation and usage of the grid system. There are several important services running in the grid middleware, such as job execution, security, information and file transfer services [67], [68]. The grid is a dynamic environment where the status of the services can be changed at any time. Each service is meant for a specific task.

- *Job execution service*: executes the user submitted jobs in a remote recourse (site). The security of the resource is guaranteed with corresponding security services. During the job execution, the job submitter shows the status of the job.

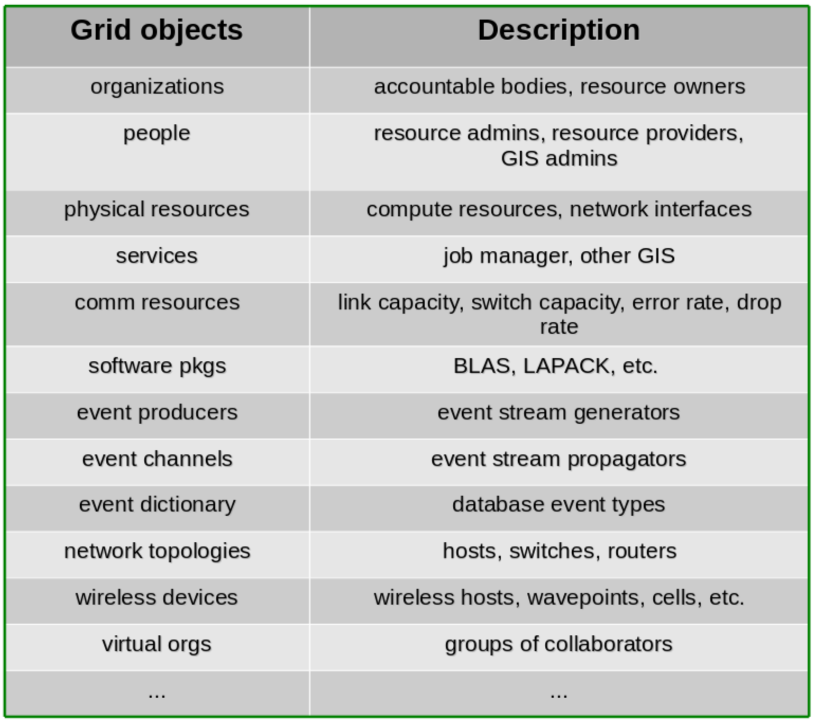
- *Grid security service*: provides the authentication and authorisation mechanism for users to access the grid resources in a secure way.

- *Information service*: provides the information about grid resources and services, such as the amount of free space in a particular computer/node, available services, running or stopped services, etc.

- *File transfer service*: provides safe and efficient data transfer from its original location to the remote node, where the task should be executed. For this, the user only needs to run a simple copy command from the Linux terminal.

2.1.2 Information Provider

Information services [67] are major and critical services in the grid middleware. They are collecting and distributing user and resource data in the grid. Information services provide detailed information about grid services that are needed for various different tasks. It maintains non-distributed and distributed information about users, software, services, and hardware that participates in a computational grid, and makes it available upon request. The grid information services are also providing data binding, discovery, lookup, and protection. In short, a grid information service (GIS) provides information about grid objects. Possible grid objects required by the information service is listed in Figure 2.8.



Grid objects	Description
organizations	accountable bodies, resource owners
people	resource admins, resource providers, GIS admins
physical resources	compute resources, network interfaces
services	job manager, other GIS
comm resources	link capacity, switch capacity, error rate, drop rate
software pkgs	BLAS, LAPACK, etc.
event producers	event stream generators
event channels	event stream propagators
event dictionary	database event types
network topologies	hosts, switches, routers
wireless devices	wireless hosts, wavepoints, cells, etc.
virtual orgs	groups of collaborators
...	...

Figure 2.8: Table of objects required for information service. Taken from [48]

In a data repository the structures of objects and the relationships between objects are described by the data model. There are four main data models: hierarchical, network, relational and object-oriented. Typical examples for the hierarchical data model are LDAP [50] and XML.

In order to provide fresh information, periodical updates of the data repositories are needed.

Information services support one of the following categories in terms of information to update.

- *Read-only repositories*: there is no standard way to update the information. Updates are usually done by other non standard interfaces, such as updates in configuration files.
- *Read-mostly repositories*: allow updates and are optimised for information reading. This can be visible in relatively slow consistency-updating protocols, or time consuming restructuring in the face of multiple additions or repositions in the data structure. For example, LDAP falls into this category.
- *Read-write repositories*: allow updates with regular read/write operations. A simple relational database management system is a read/write repository. Under this category falls all database systems (Oracle, MSSQL/MySQL, etc.).

Updates for this kind of information in GIS can only be done by an authorised person. The owner is able to perform changes to the object when the resource changes (new cluster substituted by old cluster, file server updates, etc.). The frequency of updates depends on hardware and software upgrades.

Information retrieval is done by a query interface. A query interface provides an easy way for users to obtain the necessary information. Relational and object-oriented databases support a standardised and powerful access method like the Structured Query Language (SQL) [49]. SQL queries can be complex and can contain subqueries. Hierarchical data models such as LDAP are using simplified access protocols. The queries are restricted to simple expressions which are written in a procedural language. Because of the hierarchical structure of LDAP and XML, a user must have a deep knowledge of the tree structure to write queries. The security in GIS is guaranteed by X.509 certificate.

2.1.3 Grid User Interface

The user interface (UI) [67] provides an interface for users to access grid services. The UI itself provides a command-line interface (CLI) to the middleware services of the grid infrastructure. A CLI is a cooperation between the user or client and a computer program. It is a terminal window where a user types lines of text (command-line) manually.

CLI are mostly used by advanced computer users, as they usually provide a way to control the program or the operating system.

There are a number of different CLIs, such as

- Data Catalog command-line clients and APIs,
- Data Transfer command-line clients and APIs,
- gLite I/O client [51] and APIs,
- R-GMA client [52] and APIs,

- VOMS CLI tools,
- Workload Management System clients and APIs,
- Logging and Book-keeping clients and APIs.

Each CLI has its own or specific collection of tools or commands through which a user can do different activities in the grid. For example, submit a job to the grid, delete the job, copy the file from one place to another, etc.

2.1.4 Computing Element

The Computing Element (CE) [67] is a cluster of grid resources, which is used to run jobs.

Each CE consists of three main elements, the *grid gate (GG)*, the *local resource management system (LRMS)* and the *worker nodes (WN)*.

- The *grid gate (GG)* is the interface between the cluster and a number of worker nodes. The common implementation includes an LCG grid element and a gLite grid element.
- The *local resource management system (LRMS)*, which is also called "batch system", manages the distribution of local resources (WN) in the cluster. The general implementation includes Maui/TORQUE [53] and Condor [54].
- The *worker Nodes (WN)* are the computing resources. WNs are typically desktop computers and in the grid system they are used for running the jobs.

Users submit their jobs to the grid. The GG accepts user jobs and then dispatches them to the LRMS. The LRMS takes care of the further user job execution by distributing them to the WNs. A schematic view is shown in Figure 2.9.

2.1.4.1 Portable Batch System (PBS)

Generally, the number of jobs submitted by different users are more than existing WNs. In order to solve this problem job queuing systems are needed. Each batch system has a queuing system. The batch queuing system is scheduling jobs for further execution. Typically, the job selection mechanism follows the model first-in, first-out (FIFO).

Job scheduling has two phases. The first phase is the job selection from the submitted jobs. The second phase is the allocation of memory and CPU resources according to the selected job requirements.

Different batch systems have different job schedulers. One example for the batch system is the PBS (Portable batch System) [55]. The PBS consists of four main daemon processes, such as

- *Job Server*: manages jobs and queues. The server also transfers jobs to the associated execution server.

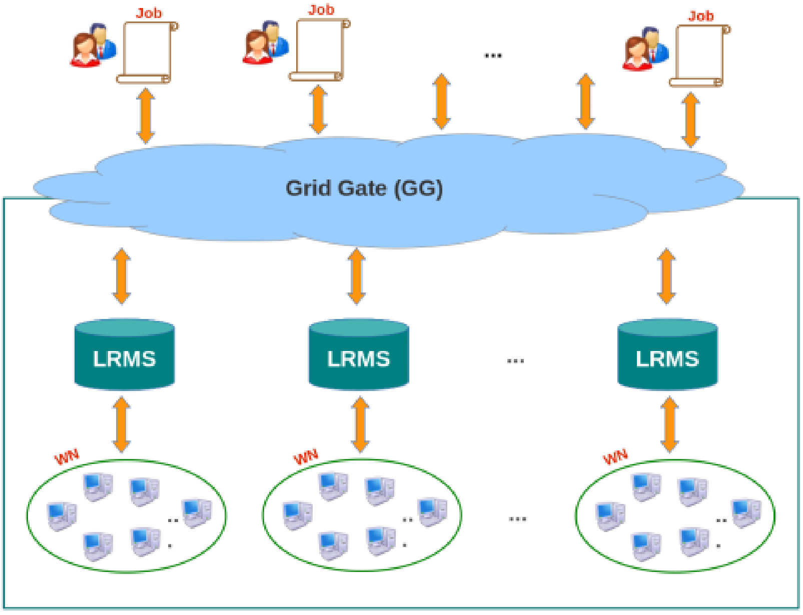


Figure 2.9: Workflow of a CE.

- *Resource Monitor*: collects the resource availability information about the host where it actually runs.
- *Job Execution Server*: is also known as the Machine Oriented Miniserver (MOM), MOM is taking care of the job execution. It also establishes limits on resource usage, monitors the job's usage, and notifies the server when the jobs are complete. Another activity of MOM is to react to the task manager requests. This means communicating with the running task and gathering necessary information. In case of system errors, log files are available in the `mom_logs` directory.
- *Job scheduler*: receives an information about the jobs, which are ready to run or are already running on the system. It also gets information about resource availability and usage.

A schematic view is shown in Figure 2.10.

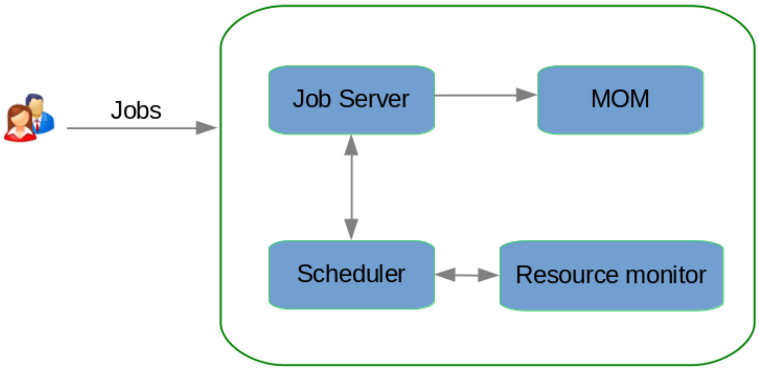


Figure 2.10: Schematic view PBS.

User jobs arrive at the Job Server, then the Resource monitor has the necessary information about the available resources. After exchanging the information between the Scheduler and the Resource Monitor and between the Scheduler and the Job Server, the jobs in the queue go to the MOM, where they start their execution.

The core functionality of the batch system is the following:

- Jobs in the Batch System are assigned/grouped in queues.
- User (job submitter) should define in advance to which queue the job will be assigned – if not, job will be assigned to the default queue.
- Queues have three main groups of attributes
  - *Limits for resources*: maximum job run time, memory or CPU core usage, etc.
  - *Priorities*: according to the scheduling strategy
  - *Security options*: who and from where is allowed to submit the job to the queue.
- Queues, hence jobs are following a particular scheduling strategy defined for a batch system in the moment of configuration.

There are several common important scheduling strategies which might be applied to the batch systems. These are the following.

- *First in, First out (FIFO) scheduling*: jobs are simply scheduled in the order in which they are submitted.
- *Preemption*: jobs get suspended in order to free resources for time-critical jobs. The state of a suspended job is stored in order to resume it later ("checkpointing").
- *Node-sets*: parallel jobs should run on the hardware with the same performance. Avoid that sub-jobs have to wait for slowest sub-job.
- *Fair share scheduling*: job priorities are calculated based on the resource usage, the submitter's group membership, etc.
- *Advanced reservations*: resources can be reserved in advance for a particular user or job.
- *Back-fill*: can be combined with any of the scheduling algorithms to allow smaller jobs to run while waiting for enough resources to become available for larger jobs. Back-fill of smaller jobs helps maximise the overall resource utilization. **D** can be filled with short jobs that have less priority than job **C** (Figure 2.11).

There are different types of batch systems, such as PBS [55], TORQUE [56], Platform LSF [57] and IBM Loadleveler [58]. Different batch systems use different CLIs as shown in Figure 2.12.

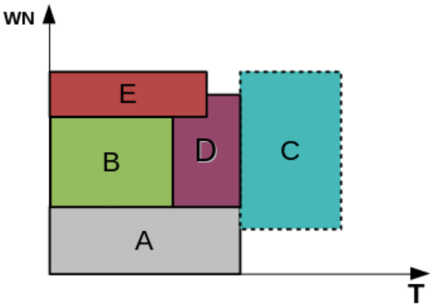
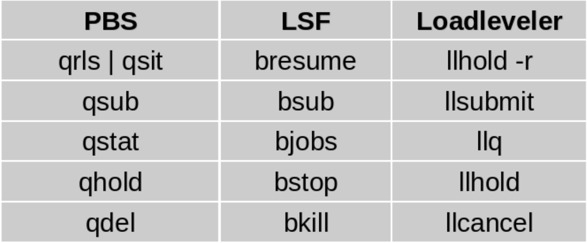


Figure 2.11: Back-fill algorithm.



PBS	LSF	Loadleveler
qrls | qsit	bresume	llhold -r
qsub	bsub	llsubmit
qstat	bjobs	llq
qhold	bstop	llhold
qdel	bkill	llcancel

Figure 2.12: Different batch systems with their CLIs.

2.1.5 Storage Element

Storage is a widely used term for the devices and data are connected to the worker nodes (WN) via input/output objects. It is a place where data are physically stored. In the grid system, files are replicated and stored on several WNs which are located in various geographical locations, in order to provide rapid multi-access capabilities. It also provides an authorisation policy that defines rules about who is allowed to modify or delete the data. Different storage resources offer different levels of Quality of Service (QoS) [59].

The storage hardware is controlled by the software. The storage resource is the combination of the storage hardware and supported software.

A storage element (SE) [67] has an interface for accessing grid storage through different grid protocols, such as the Storage Resource Management protocol (SRM) [60], the Globus GridFTP protocol [65] and others. The SE also allows grid users to store and manage files/data with the space assigned to them.

In order to make storage available for usage different services are needed. There are three main interfaces used for the storage resource shown in Figure 2.13.

- *Data Access*: rfio [62] , dCap [67] protocols
- *Data Transfer*: GridFTP [65] protocol
- *Storage Management interface*: SRM [60] protocol

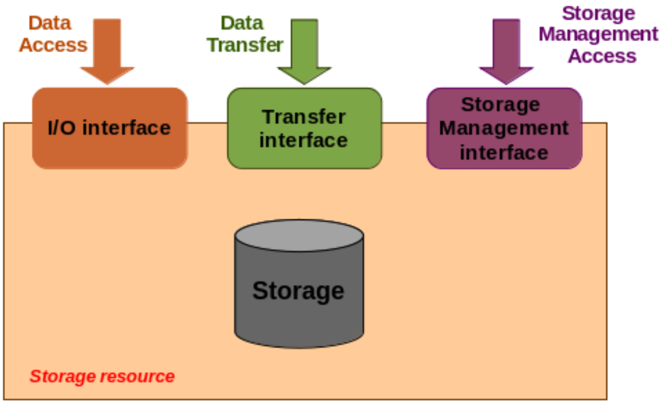


Figure 2.13: Storage resource.

2.1.5.1 Data Access (rfio, dCap) Protocols

rfio [62] and dCap [67] are protocols that provide access to hierarchical mass storage systems like Castor and dCache. The rfio protocol was designed to transfer files in a trusted environment with a minimum of overhead. rdio does not use a secure authentication while other protocols such as GridFTP and SRM use a secure authentication mechanism. rfio provides a full POSIX access mechanism to the remote files via adapted *open, read, write* and *close* commands. rfio has a problem with many parallel requests.

The dCap protocol file supports the regular file access functionality. It offers POSIX *open, read, write, stat, close* operations as well as standard file system namespace operations. dCap supports security mechanisms where security protocols are implemented.

2.1.5.2 Data Transfer (GridFTP) Protocol

GridFTP [65] is a secure and reliable data transfer protocol. GridFTP is based on FTP (File transfer protocol) and fulfills requirements of grid systems.

As an extension of FTP, GridFTP has the following main features:

- *Authentication*: GridFTP integrates with the Grid Security Infrastructure that provides authentication and encryption to file transfers with user-specified levels of confidentiality and data integrity.
- *Parallel streaming*: GridFTP supports the usage of parallel streaming and multinode transfers.
- *Basic transfers*: GridFTP provides data transfers between two worker nodes. An example is shown in Figure 2.14. Site A runs a GridFTP server and holds data that will be transferred to Site B, which runs a GridFTP client.
- *Third-party transfers*: GridFTP performs transfers between two worker nodes, mediated by a third host (Figure 2.15). This allows a user or application at one site to initiate, monitor and control a data transfer operation between two other sites.

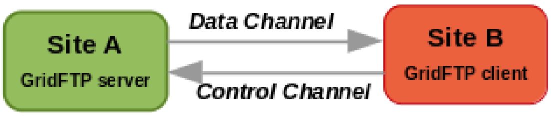


Figure 2.14: Basic GridFTP transfers.

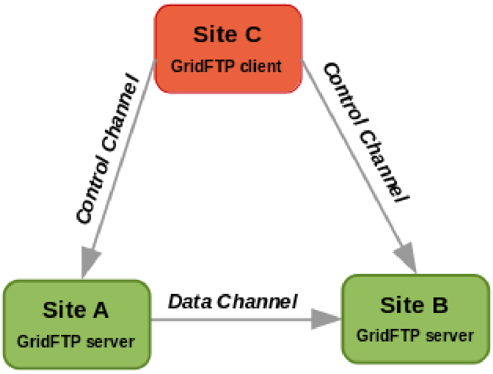


Figure 2.15: GridFTP third-party transfer.

- *Striped/Partial transfers*: GridFTP supports striped and partial data transfers between hosts.

In GridFTP there are two different communication channels, the *data* and *control* channels.

The *data channel* is the communication link with high-bandwidth over which the data transfers from one place to another.

The *control channel* is a low-bandwidth TCP link over which commands and responses flow, which is encrypted and integrity-protected by default. The direction of the arrow depends on where the GridFTP client and the GridFTP server are located.

The Globus Toolkit provides the most common usage of the GridFTP protocol. It provides server, client CLI tools (*globus-gridftp-server*, *globus-url-copy*) [63] and a set of developed libraries for the clients.

Another usage of the GridFTP protocol is developed by NCSA and provides the client CLI (UberFTP [64]).

GridFTP is defined by the Global Grid Forum Recommendation GFD.20 [65], GFD.47 [66].

2.1.5.3 Storage Resource Management (SRM)

The Storage resource manager (SRM) [60] is managing an associated storage by using different techniques such as cache management, dynamic space allocation and reservation for data transfers, staging of data from slow to fast storage in a hierarchical storage system and scheduling storage system requests. A SRM is typically managing two types of shared resources, files and space. Its main function is to allocate space for a user dynamically, check the permission of a user to use that particular space, assign the space to the user for the given time period according to its policy and release the space when the user requests its release or when the time is expired. A SRM also supports file replication between SEs, which can be considered as third-party transfers. This is different from GridFTP third-party transfers as SRM is using credential delegation between the client and the local SE.

The SRM protocol version 2.2 [61] functionality can be grouped in the following five categories: space management functions, data transfer functions, directory functions, permission functions and discovery functions.

Space management functions - allow the user to reserve, release and manage spaces for some defined time period. The main space management functions are: *srmReserveSpace*, *srmUpdateSpace*, *srmReleaseSpace*, *srmChangeSpaceForFiles*, *srmExtendFileLifeTimeInSpace*.

Data transfer functions - allow the user to manage, retrieve, copy, data from/to the storage. The main data transfer functions are: *srmPrepareToGet*, *srmBringOnline*, *srmPrepareToPut*, *srmPutDone*, *srmCopy*, *srmReleaseFiles*, *srmAbortRequest/srmAbortFiles*, *srmExtendFileLifeTime*.

Directory functions - allow users to manage the SRM namespace for creating, moving and deleting directories and files as well as allow users to put files with different storage requirements in a single directory. Changing the space characteristics of a file does not change its position in the directory but only its assignment to the corresponding space.

Permission functions - allow users to assign read and write privileges on a specific file to other users. Such functions allow client applications to specify ACLs as well, wherever supported by the underlying storage service.

Discovery functions - which allow applications to query the characteristics of the storage system behind the SRM interface and the SRM implementation itself.

The practical example is shown in Figure 2.16.

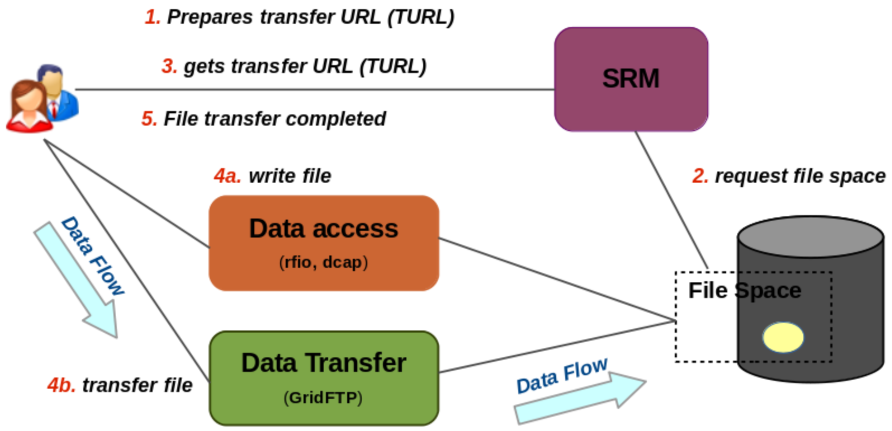


Figure 2.16: An example of copying a file to the grid storage.

A client or user requests a space reservation on a grid storage. After getting the confirmation, the client/user copies the file to the storage. As a last point, a notification is sent to the SRM that the client/user completed the file transfer for the allocated space.

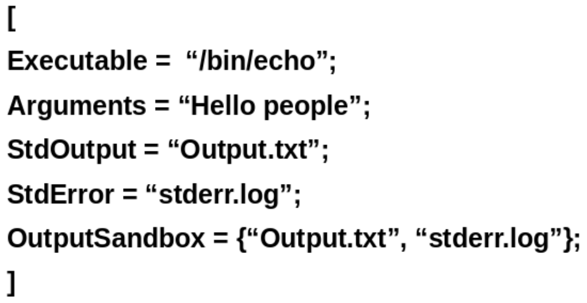
2.1.6 Workload Management System (WMS)

The grid system provides a large variety of complex services such as the data management services, the security services, the information services, etc.

The job scheduling in a grid systems is a very well-known research topic. Workload Management System (WMS) [67] is an architecture for the distributed scheduling and resource management in a grid. Usage of the WMS is very important for the grid systems.

WMS consists of several main components.

- *Job preparation*: includes specific information about the job, such as job characteristics or attributes and job requirements. The job description uses the Job Description Language (JDL) [67]. JDL defines the set of job attributes. Later, WMS consider this information for the scheduled decision making.
- *Job submission*: is the actual JDL file submission process. There are several steps that each user should follow in order to submit the job.
  - the user needs to login to the UI.
  - the user types *grid-proxy-init* command with the corresponding certificate password in order to get a valid Globus proxy certificate.
  - The user should make up his/her own JDL file. An example is shown in the Figure 2.17.



```
[
Executable =  “/bin/echo”;
Arguments = “Hello people”;
StdOutput = “Output.txt”;
StdError = “stderr.log”;
OutputSandbox = {“Output.txt”, “stderr.log”};
]
```

Figure 2.17: JDL file example.

  Submit the JDL file to the grid, for example *glite-wms-job-submit -o jId.txt job.jdl*. The *-o jId.txt* flag is used to store the job identifier in a file called *jId.txt* for later reference.

- *Job monitoring and control*: provides information about the job. Allows users to modify the job, such as edit the content of the JDL file, delete the file or just check the status of the job.

The WMS manages job submission and cancellation from the user interface via the network. It is an intermediate service that schedules jobs in the most efficient way.

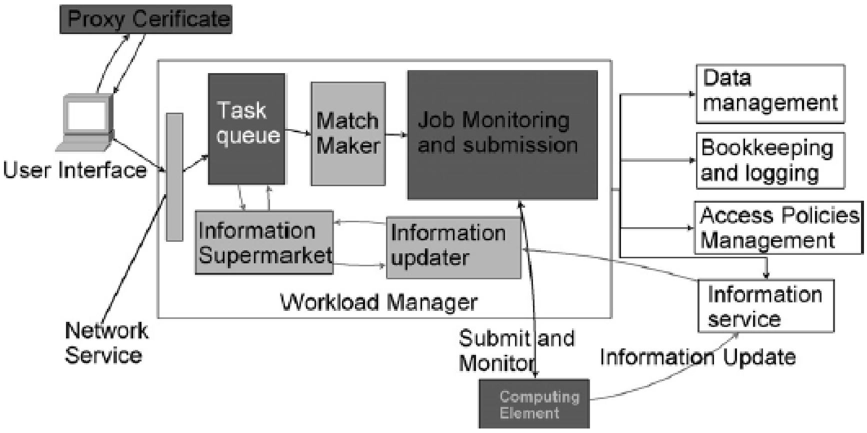


Figure 2.18: Structure of WMS. Taken from [67].

The WMS workflow is shown in Figure 2.18.

The workload manager (WM) is the main component in the WMS. It includes task queue, information supermarket, information updater, matchmaker and job handler (described below).

This component mainly takes care of the optimal job allocation and the wrapping of the job before submitting them to the computing element. The process of job allocation and wrapping is done by several grid services, such as the data management, the bookkeeping and the information services.

2.1.6.1 Task Queue

One of the WM elements is *task queue*. It is responsible for handling the user job requests and also takes care of the user authentication by digital proxy certificate. Once the job is submitted or cancelled, it gets interpreted according to the JDL attributes and after it passes or does not pass to the task queue. After the job reaches the task queue it stays in a queue for later processing by the matchmaker. If there is no suitable resource available for scheduling the job returns to the task queue and waits for processing.

2.1.6.2 Matchmaker

The next element of the WM is the *matchmaker*. This element is responsible for two main things: for the job processing and for matching them with the available resources. The job processing includes user access policies, requested resources and job privileges. Available resource information is provided by the *information supermarket*. The last provides an up-to-date information about the grid resources. Putting these two things together, the matchmaking process cares about the optimal job scheduling in accordance to the available resources in the grid. After this process, the actual job passes to the next stage, which is the actual job submission to the grid system.

2.1.6.3 Information Supermarket

The next element of the WM is the *information supermarket*. This element stores information about the grid resources. The information gets updated constantly via the *information updater*.

2.1.6.4 Information Updater

The next element of the WM is the *information updater*. This element is responsible for the up-to-date information stored in the *information supermarket*. Up-to-date information is provided by the top-level BDII server (see Section 3.2.2) of the virtual organization.

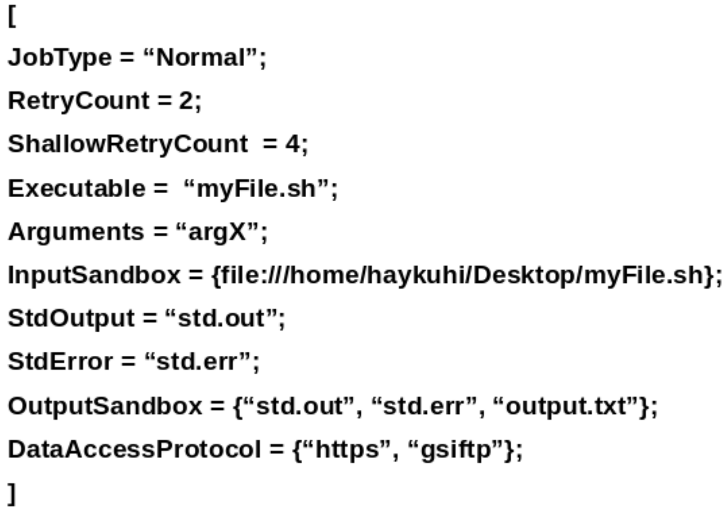
2.1.6.5 Job Handler

The last element of WM is the *job handler*. The *job handler* is responsible for the job packaging, job submission/removal and for the job monitoring. Jobs first get packed and then executed into the particular resource. Submitted jobs are actively monitored by the log monitor.

2.1.6.6 Job Description Language (JDL)

The *Job Description Language (JDL)* [67] is a high-level language used for making grid jobs.

A JDL file example is shown in Figure 2.19.



```
[
JobType = “Normal”;
RetryCount = 2;
ShallowRetryCount  = 4;
Executable =  “myFile.sh”;
Arguments = “argX”;
InputSandbox = {file:///home/haykuhi/Desktop/myFile.sh};
StdOutput = “std.out”;
StdError = “std.err”;
OutputSandbox = {“std.out”, “std.err”, “output.txt”};
DataAccessProtocol = {“https”, “gsiftp”};
]
```

Figure 2.19: JDL file example.

JDL relevant attributes are the following.

- *JobType*: normal, sequential, parallel.
- *RetryCount/ShallowRetryCount*: used for the job auto-submission and the maximum number of times that job can be resubmitted to the system.

- *Executable (mandatory)*: should mention the filename of the executable file.
- *StdInput, StdOutput, StdError (optional)*: standard input/output/error of the job.
- *InputSandbox (optional)*: stores the command path for the further job execution.
- *OutputSandbox (optional)*: contains executed job output files, that are typically std.out and std.err files.

Chapter 3

Grid Middleware

3.1 Introduction

The grid system includes storage resources, network resources and computational resources such as servers, supercomputers and personal computers, which are located in different geographical locations. In order to use grid resources effectively some grid middleware is needed, which provides access to the grid resources and facilities.

The grid middleware is a shared, extensible and flexible environment. It is a set of APIs, software and different protocols that allow the creation of the grid system and its usage.

Because of the complexity of the grid architecture and the diversity of its functions, the interaction between middleware and other software is also required.

The grid architecture [30] is a very complex architecture because of its variety of functions. The communication between other software and grid middleware is required.

The grid architecture has a layered structure, shown in Figure 3.1.

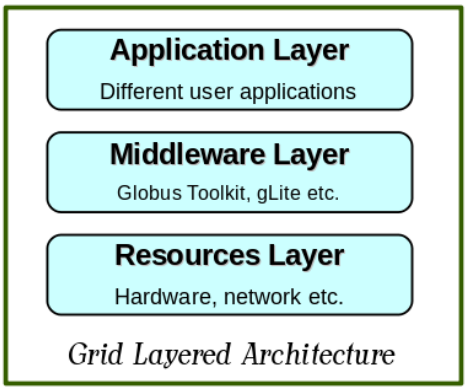

Application Layer
Different user applications
Middleware Layer
Globus Toolkit, gLite etc.
Resources Layer
Hardware, network etc.
Grid Layered Architecture

Figure 3.1: Grid layered architecture.

The bottom layer of the grid architecture is the grid fabric, which includes hardware, network and all other necessary resources.

In the top level of the grid architecture is the application layer, which includes different user applications and portals.

In the middle layer between bottom and top layers is the grid middleware itself, which enables a connection between the hardware and the user applications. In other words, it provides software, protocols and APIs for the grid systems.

There are different grid middleware, such as Globus Toolkit [33], gLite [69], UNICORE [70], etc. At CERN, for the LHC experiments we are using the gLite middleware (see Section 3.2).

3.2 gLite Middleware

The gLite middleware [69], [71] was developed by the Enabling Grids for E-sciencE (EGEE) [72] project in order to provide all necessary tools for resource sharing. It is a big collaboration between scientists and engineers from all over the world. It is a very reliable and highly accessible middleware.

The main components of gLite are the following (Figure 3.2):

- Access methods
- Security services
- Information and monitoring services
- Workload Management services
- Data Management services

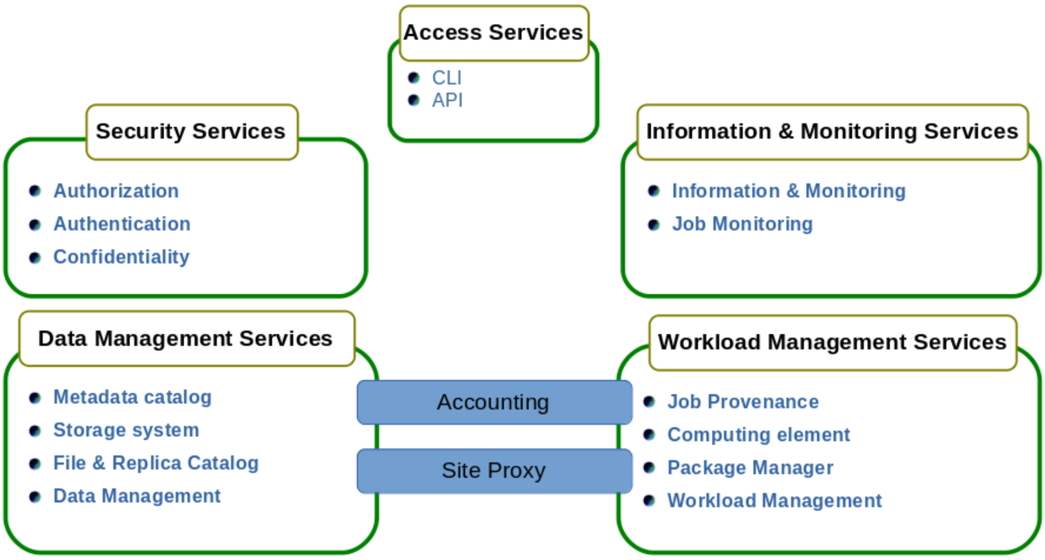


Figure 3.2: gLite services [71].

Access Services are used for providing users the access to the grid services according to their rights. All gLite services can be accessed via CLI or application programming interface (API).

Security Services include authentication, authorisation and auditing services that are used to identify different objects (users, systems and services) and allow or deny the access to grid services and resources. Security services also provide the confidential data and a site proxy, which is used for controlling the access via network.

Information and Monitoring Services are meant to extract and publish information for monitoring purposes. Published data includes the time and date information, as well as the identity of the publisher and the location from where it was published. The information is not allowed to be modified as data inconsistency may occur. There are also different ways of cleaning out the old data. As a last point, there are well-defined authorisation schemes to guarantee that users can only read or write data according to their privileges.

Workload Management Services are related to the job management/execution in the computing element (CE), the job provenance, the accounting and the package manager services. The accounting is a distinctive service as it takes into account not only the computing, but also the network and the storage resources.

Different workflows for the job management exist, for example *push* and *pull* models.

- The purpose of the *push* model is for the WMS to decide where to execute a job taking into account its availability, its location and the characteristics of the worker nodes (WNs) in the CEs.
- The purpose of the *pull* model is for the WMS to collect jobs in a task queue and then the CEs request jobs by taking into account its availability, its location and the characteristics of the worker nodes (WNs) from the WMS.

Data Management Services include three main components, such as the storage elements (SEs), the file catalogues and the file transfer service. The SE (see 2.1.5) is a gLite component that stores data for later retrieval by other gLite applications or by grid users. Here, files are replicated and stored in different locations in order to provide quick multi-access capabilities. Data consistency is also supported. Data are stored in files more often than in relational databases [73].

In a distributed environment, there are many file replicas that are stored in different geographical locations. The middleware provides the functionality of replica management. Before performing the actual file transfer operations in the SEs, files have to be located and identified. The identification of files on the storage elements is done via different identifiers ([69]) such as the Grid Unique Identifier (GUID), the logical file name (LFN), the Storage URL (SURL) and the Transport URL (TURL). The LFN is the key, where users write the actual locations of the files. The replicas are identified by SURLs. SEs extract necessary data by contacting the SURL, which is unique for each replica. The access to the files is controlled by the Access Control Lists (ACL).

The Data Management Systems provide all necessary interfaces to the users for data placement in a distributed environment, where the transmission of the data is scheduled as well as the jobs in the WMSs.

The main gLite middleware components are the following

- UI - User Interface,
- CE - Computing Element,
- SE Storage Element,
- WN - Worker Node,
- WMS - Workload Management System,
- VOMS - Virtual Organisation Membership Service
- LB - Logging and Bookkeeping,
- MDS - Monitoring and Discovery Service,

- LFC - Logical File Catalog,
- BDII - Berkley Database information Index.

The details about the above listed gLite components are covered in chapter 3. The Figure 3.3 gives an overview over the gLite middleware.

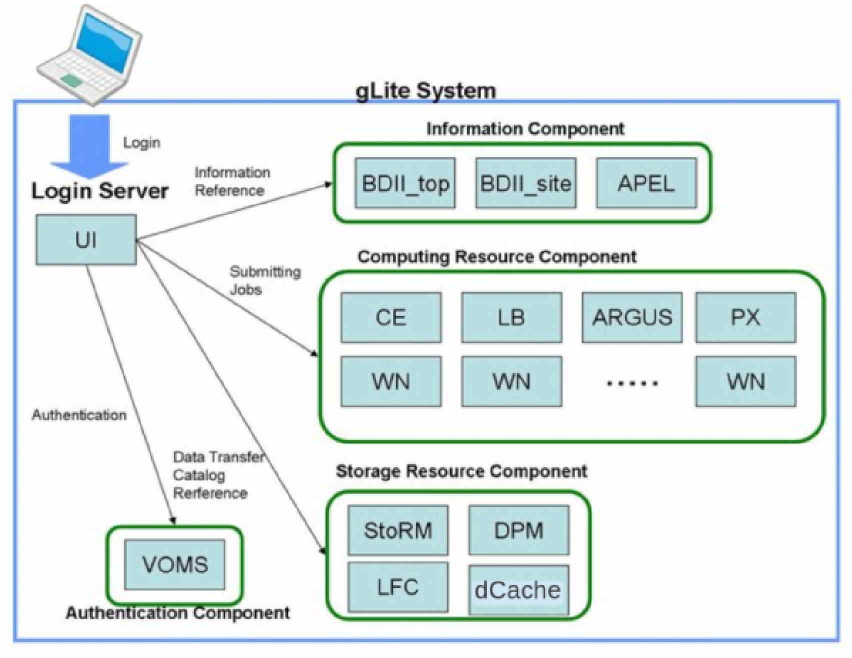


Figure 3.3: gLite components.

3.2.1 gLite-VOMS

gLite remote services usually act on behalf of the user, which is called delegation. This means that the user should authorise the remote service before he can do any activity in the grid. The authorisation process proceeds by using the X.509 user certificate. The delegation of the certificate is called the proxy. Before doing any activities in the grid, users should login to the grid. This is done by creating the temporary proxy certificate using the original X.509 user certificate. The proxy certificate has a limited lifetime. For security reasons the proxy lifetime is only 12 hours.

In gLite, the user authentication is done by using the X.509 certificate and the authorisation is done by the VOMS attribute (see Section 2.1.1.6). An authorisation is based on the Access Control Lists (ACL), which guarantees data access only by their owners. Through the VOMS provided tools, users are allowed to generate their own proxy credentials based on the VOMS database content and their own X.509 proxy credentials. The gLite security mechanism is shown in Figure 3.4.

The creation of proxy certificates is very simple. It requires the execution of the *voms-proxy-init –voms <VO>* command [69], where the VO is the name of the virtual organisation. In our case we use ATLAS as the VO, an example is shown in Figure 3.5. The X.509 certificate password is required for the creation of the proxy certificate.

In this example the proxy is located in the */tmp* directory.

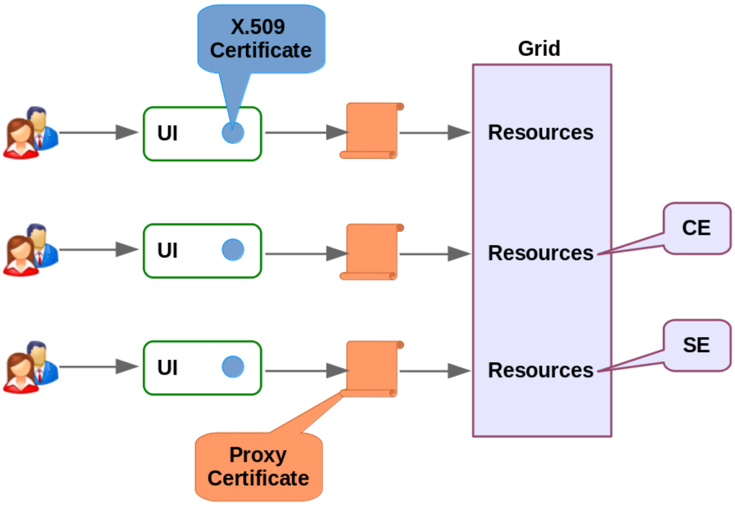
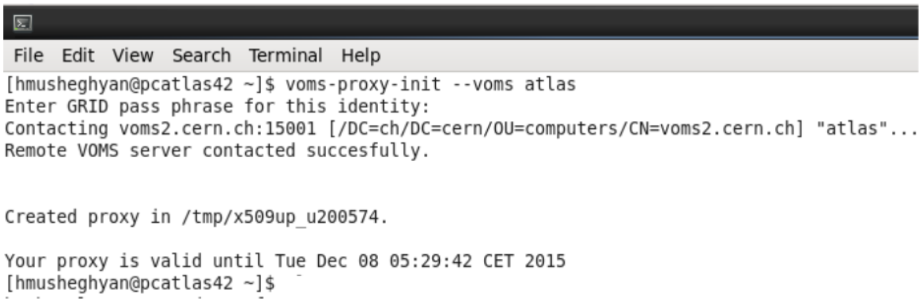


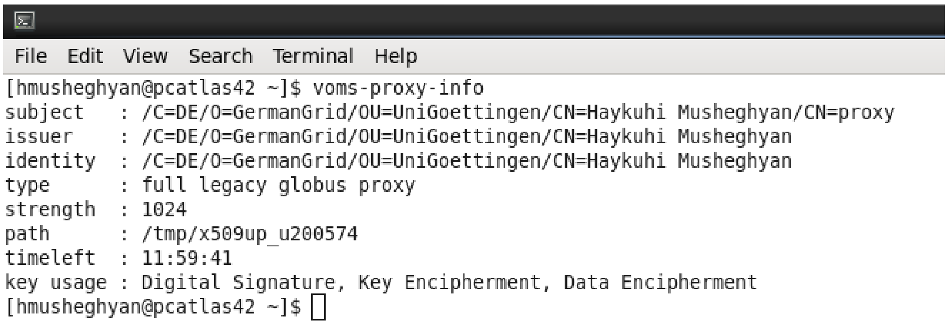
Figure 3.4: Overview of certificates in gLite security mechanism.



```
[hmusheghyan@pcatlas42 ~]$ voms-proxy-init --voms atlas
Enter GRID pass phrase for this identity:
Contacting voms2.cern.ch:15001 [/DC=ch/DC=cern/OU=computers/CN=voms2.cern.ch] "atlas"...
Remote VOMS server contacted succesfully.


Created proxy in /tmp/x509up_u200574.

Your proxy is valid until Tue Dec 08 05:29:42 CET 2015
[hmusheghyan@pcatlas42 ~]$
```

Figure 3.5: An example of an executed *voms-proxy-init* command.

In order to have more information about the created certificate, the *voms-proxy-info* command [69] should be executed (Figure 3.6).



```
[hmusheghyan@pcatlas42 ~]$ voms-proxy-info
subject   : /C=DE/O=GermanGrid/OU=UniGoettingen/CN=Haykuhi Musheghyan/CN=proxy
issuer    : /C=DE/O=GermanGrid/OU=UniGoettingen/CN=Haykuhi Musheghyan
identity  : /C=DE/O=GermanGrid/OU=UniGoettingen/CN=Haykuhi Musheghyan
type      : full legacy globus proxy
strength  : 1024
path      : /tmp/x509up_u200574
timeleft  : 11:59:41
key usage : Digital Signature, Key Encipherment, Data Encipherment
[hmusheghyan@pcatlas42 ~]$ 
```

Figure 3.6: An example of an executed *voms-proxy-info* command.

Here we can see the certificate type, which is proxy, the issuer name, the validity and the location of the proxy certificate and key usage.

3.2.2 gLite-BDII

The information services [67] provide information about the status of grid resources. The role of the information services in the grid are very important (see Section 2.1.2).

The functionality of the information service in gLite is shown in Figure 3.7.

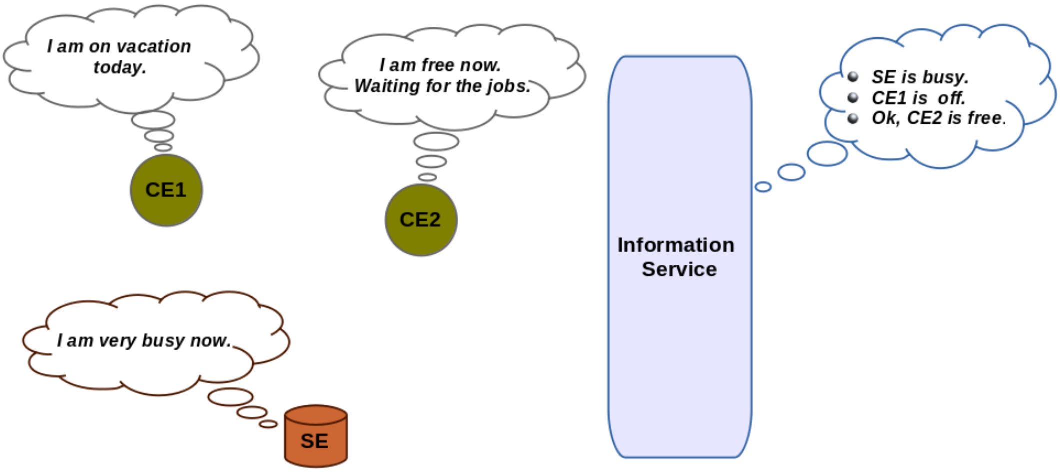


Figure 3.7: Functions of information service in gLite.

There are two information service mechanisms in gLite:

- *Relational grid monitoring architecture (R-GMA)*: used for accounting, monitoring and publishing of user level information. R-GMA uses the servlet technology for the intercommunication in gLite. By using communication protocols, the data transmissions between each other is done in an efficient way. This technology also allows easy and quick adaptation to other web services.

- *Globus monitoring and discovery service (MDS)*: gLite version 3.1 inherits the main concepts of the Globus MDS. The MDS is used for the resource discovery and the status publication. It has a tree structure for gathering information (Figure 3.8).

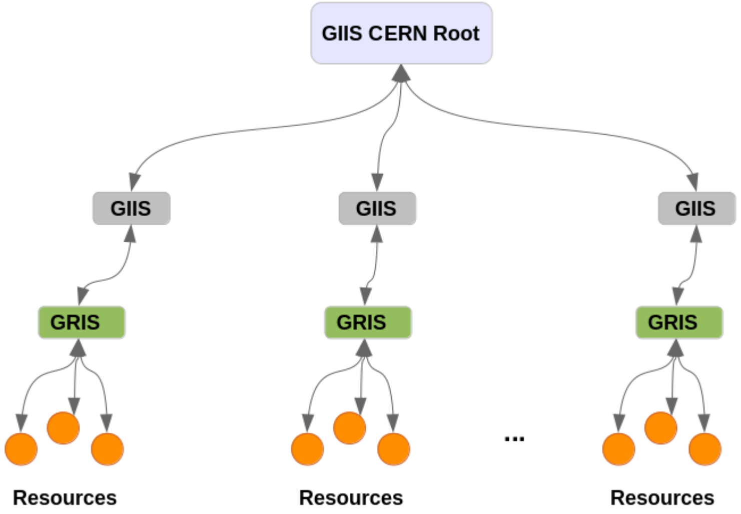


Figure 3.8: Hierarchical tree structure of MDS.

At the bottom level are individual resources (one for each CE or SE) and information is provided by the information provider. This information is then collected and published by the grid resource information server (GRIS). Afterwards, the information is passed to the next level, which is the Grid Index Information Service (GIIS). For each site, there is only one GIIS that collects information from the GRIS', located one level below. This service also caches the information according to its validity time.

The Berkeley Database Information Index (BDII) [74] includes standard LDAP databases [50] (two or more). The BDII is on the top of the MDS structure (Figure 3.9).

Each BDII is monitoring a certain amount of the GIIS. The BDII contains all the information about the available resources at the sites. Finally, at the highest level of all the BDIIs is the VO.

In this level there is an overall BDII that merges different BDIIs in the same VO together. It is a server where all valuable information concerning the VO is stored.

3.2.3 gLite-UI

The gLite user interface (UI) [69] is an interface for users to access grid services. It is a collection of different CLI clients and APIs, such as the data catalog, the data transfer, the I/O, the R-GMA, the VOMS, the WMS and the Logging and Book-keeping. Through the CLI tools, users can perform several grid operations like:

- submit and cancel jobs in the grid,
- check the status of jobs in the grid,
- retrieve the output of finished jobs from the grid,

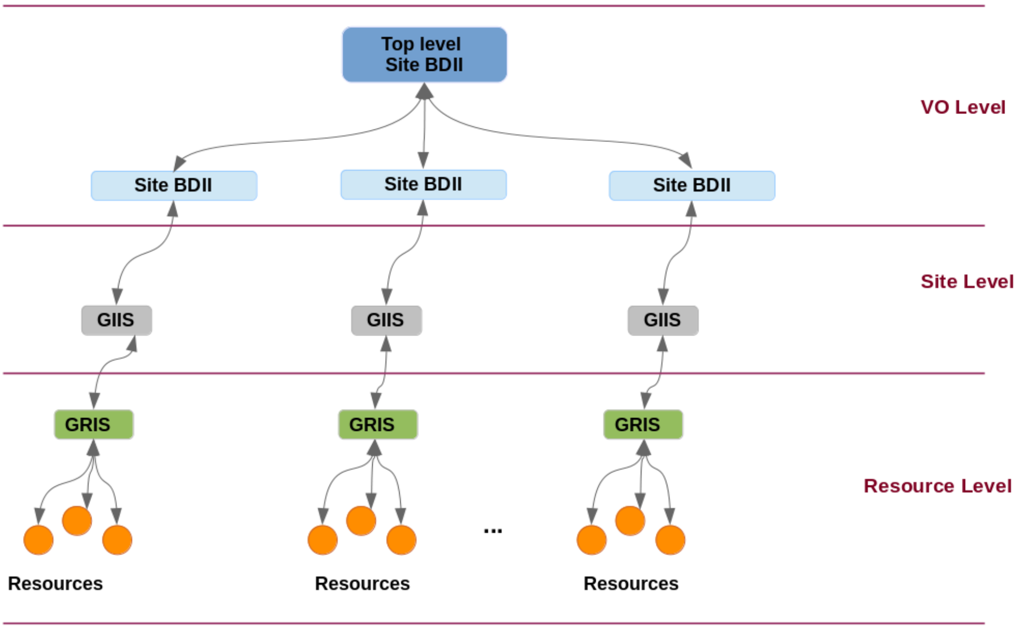


Figure 3.9: Hierarchical tree structure of the MDS and the BDII.

- copy, delete and replicate files from the grid,
- get information about the various resources available in the grid.

The available APIs allow also to develop grid-enabled applications.

3.3 CREAM Computing Element

A CE has a complex structure. It represents an interface to a large computing resource managed by a Local Resource Management System (LRMS), such as PBS and LSF. Additionally, CEs are also providing a grid user authentication and authorisation, accounting, fault tolerance and improved performance and reliability to these batch systems.

The Computing Resource Execution and Management (CREAM) [75] is an efficient job management system used for grid environments. It is the CE implementation in the gLite middleware. The main goal of CREAM is to provide a robust, simple and efficient service for grid job operations. CREAM is written in the Java programming language and it uses Tomcat as a server. It includes an interface that is based on Web Services and is very flexible and compatible with different clients that are written in different programming languages. CREAM is widely used in different areas such as bio-medicine, physics, astrophysics, high energy physics, finance, etc.

The main functionality of CREAM is the job submission. CREAM supports several activities, such as

- the execution of batch and parallel jobs (MPI) [76],
- the transfer of the Input Sandbox (ISB) (a set of files needed for the job execution) to the executing node, from the user WM or from the grid storage server,

- job management operations, like the job status, job cancellation, job posting and job purging. Users are also allowed to suspend or restart submitted jobs.

Users submit their jobs, which are using JDL as the script language to the CREAM CEs. The JDL used for CREAM and WMS is the same (see Section 2.1.6.6). This means that gLite WMS and CREAM can be integrated together. CREAM can work as a grid component of gLite WMS. User jobs that are submitted to the gLite WMS can be forwarded to CREAM-based CEs for further execution. The CREAM-WMS integration works together with a separate module named Interface to Cream Environment (ICE) (Figure 3.10).

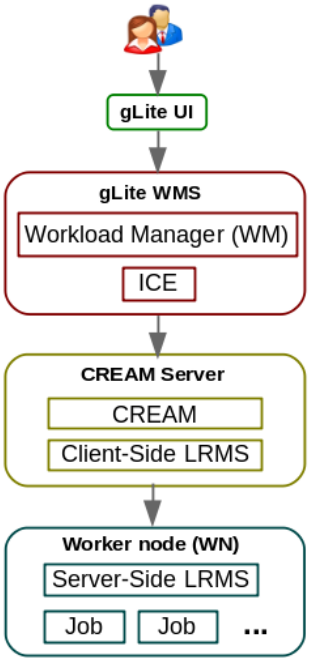


Figure 3.10: CREAM-WMS integration.

ICE receives job requests from WMS and then uses the corresponding CREAM methods to perform the request. ICE is also responsible for the monitoring of job status and to take the appropriate decision if the job status is changed.

3.4 The dCache Storage Element

One of the most used storage management systems is the so-called dCache system [77]. dCache is a storage management system developed at DESY in collaboration with a group at FermiLab since 2003.

The dCache is a highly sophisticated storage management system used within the WLCG. It is a very effective system that is able to deal with several hundreds of terabytes of data. dCache stores and provides a huge amount of data that are distributed and located in different geographical locations. It provides a single virtual file system with different standard access methods and disk pool management system with or without a tape backend. The dCache also has an automatic disk load balancing system.

The dCache system is written in Java and it uses standard open source technologies. The main functionality and features of the dCache system are the following:

- Hierarchical Storage Management
- Manages the enormous amounts of data (no hard limit, tested in the petabyte range)
- Modular design and portability, highly configurable and customisable system
- Uses filesystems , such as ext4, XFS or btrfs to store the actual data on the storage-nodes
- Authentication is based on the usage of X.509 certificates, username+password and Kerberos
- Supports access protocols, such as DCAP/gsiDCAP (native protocols), NFS, GridFTP, HTTP/WebDAV, SRM
- Scriptable administration interface with terminal-based and graphical front-ends
- XML information provides detailed live information about the storage system
- Web-interface shows live summaries of the most important information
- Powerful cost calculation system that allows the control of data flow (from and to pools, between pools and also between pools and tape)
- Load balancing and performance tuning by "hot replication" (via cost calculation and replicas created by pool-to-pool-transfers)
- Garbage collection of replicas, depending on their flags, age, etc.
- Space management and support for space tokens

The user reads/writes data to the cluster as shown in Figure 3.11.

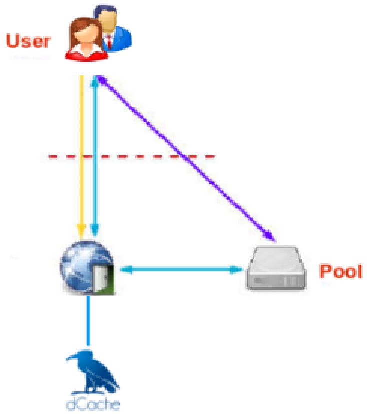


Figure 3.11: User transfers data to the cluster.

First the user with the valid X.509 certificate connects to a door with the desired protocol. Then dCache determines the pool from which the data should be read or where the data should be written to. Depending on the protocol and configuration the user request of the actual data can happen either directly between the client and the pool or indirectly via the door.

3.5 Description of GoeGrid

GoeGrid [78] is a grid computing resource center located in Goettingen.

GoeGrid is mostly used by university scientists who are doing research in different departments such as high energy physics, computer science, theoretical physics, astrophysics and biomedicine. Besides being a grid center, GoeGrid is also a Tier-2 for the ATLAS experiment within WLCG (see Chapter 4). The GoeGrid Tier-2 center is located in the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG).

The GWDG hosts the GoeGrid Tier-2 and serves as a main computer center for the University of Goettingen and the local Max-Planck-Institutes. Besides that, it also maintains the hardware, the network infrastructure and the water-cooling of the racks.

The performance and availability of the GoeGrid Tier-2 center is a complex task and is done by having necessary hardware and corresponding software administration tools. For different departments, the requirements in terms of hardware and software setup are different. Therefore, every department has its own contribution into the center. The benefits are an efficient use of computer and human resources.

The ATLAS detector has approximately 100 million readout channels to detect collision events. This leads to a data rate of 1 TB/s. The GoeGrid cluster provides 634 CPU-s and 3496 cores.

All communities that are using GoeGrid agreed on the usage of a common software setup. Setup means requirements of the various applications. The structure of GoeGrid is shown in Figure 3.12.

CE and the batch system

In GoeGrid, the used batch system is called TORQUE [53] and the job scheduler is MAUI [53].

TORQUE is an open-source resource manager used to control batch jobs and distributed computing resources. It is an advanced product in terms of scalability, reliability, functionality and it is currently used by thousands of leading academic, government and commercial sites throughout the world. TORQUE is one version of PBS (see Section 2.1.4.1) and has similar functionality.

MAUI is a job scheduler mainly used for clusters and supercomputers. It is supporting multiple scheduling policies, dynamic priorities, reservations, and fair-share capabilities. In GoeGrid, taking into account the shared resources between several scientific groups, the used scheduling algorithm is the fair-share scheduling algorithm [79]. The fair-share algorithm insures the control of the shared resources and allows site administrators to set system utilisation targets for users, groups, accounts, etc.

GoeGrid uses CREAM (see Section 3.3) as a computing element manager. For security reasons, GoeGrid has two different CE instances which are running CREAM. The CREAM component takes care of the communication with the batch system (PBS/ TORQUE). It accepts two types of jobs: regular batch jobs and parallel jobs. For regular jobs there are no special software requirements, while for parallel jobs the MPI library is needed as it allows inter-process communication.

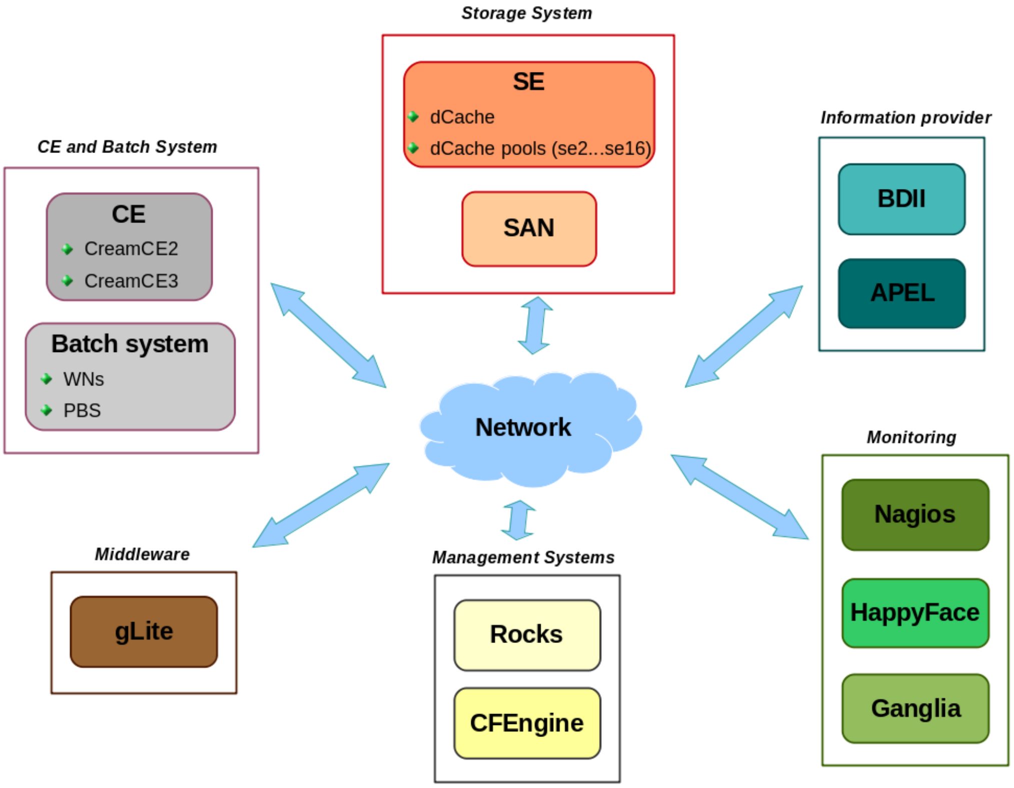


Figure 3.12: GoeGrid structure.

Storage system

The GoeGrid storage system is based on dCache (see Section 3.4). It allows the cluster to have a single storage space with a general file system. The storage space is divided into several space tokens such as *GOEGRID_DATADISK*, *GOEGRID_DETINDET*, *GOEGRID_LOCALGROUPDISK*, *GOEGRID_PHYS-EXOTICS*, *GOEGRID_ PHYS-HIGGS* and *GOEGRID_SCRATCHDISK*. Each of these space tokens has its own meaning and is meant for specific usage. This is an official ATLAS Distributed Computing (ADC) requirement. The access to the space tokens is guaranteed by the SRM protocol.

In addition, the SAN [80] mass storage system of the GWDG provides approximately 30 TB of storage space. The SAN is maintained by the GWDG.

Information provider

The calculation and management of the available computing resources is very important as it leads to an effective resource management. For each ATLAS site the computing resource information is needed to be open within the collaboration. In GoeGrid, this task (the CPU accounting information) is done by the *APEL (Accounting Processor for Event Logs)* [81] service.

The APEL is the fundamental tool for accounting and publishing the CPU usage information in the grid infrastructure. The APEL parses the log information of the batch system (in the GoeGrid case it is PBS/TORQUE) to produce CPU job accounting records in the grid systems. More accurately, it collects the complete information about the usage of computing resources by user jobs. APEL then publishes accounting data into an accounting record repository at a Grid Operations Center (GOC) (in the GoeGrid case, the center is located at Rutherford Appleton Laboratory (RAL) [82]).

APEL consists of three main components: the APEL parser, the APEL core and the APEL publisher. The APEL parser gets the needed information (submitted and finished jobs status) from the batch system via the computing element. The APEL core stores aggregated information into the APEL database. The APEL publisher gets the accounting data from the APEL database and publishes them in the GOC.

The *BDII* service (see Section 3.2.2) provides detailed information about the site and its available resources. The GoeGrid BDII provides information about the available CPU cores and running grid services required by the ADC management.

Monitoring

Monitoring of such a big cluster needs to be done by an efficient multi-level monitoring system. This means the usage of different tools for monitoring the status of hardware and software. For GoeGrid we use tools, such as Nagios [83], HappyFace (see chapter 6), Ganglia [86], etc.

Nagios [83] is a professional open-source monitoring tool for the entire IT infrastructure. Nagios provides a notifying system for the servers, WNs, switches and other possible hardware. A Nagios GoeGrid web interface is shown in Figure 3.13.

In this example, the Nagios monitoring tool displays the status of apel, bdii and the pbs hosts that are running in GoeGrid. It shows the current load of the hosts, the status of different partitions, processes, etc. and the last check time, the duration of the test, the number of attempts and status information.

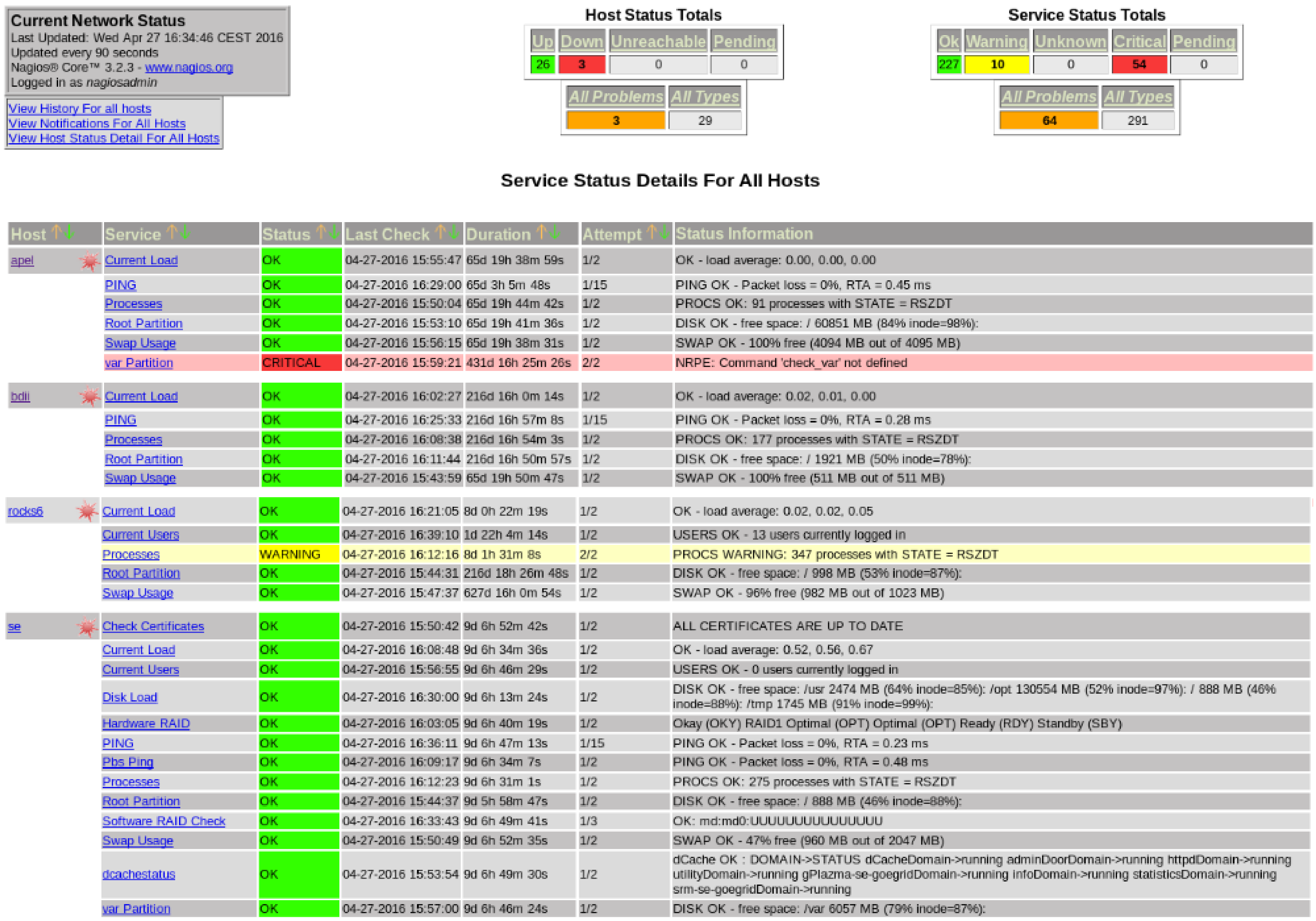
If the test of some particular service fails, then the whole line is marked in red and the status changed to CRITICAL. More detailed information about the hosts or services, can be seen by following the structure of the site.

Management Systems

To provide available and reliable services for the grid site, the first part to focus on is the management and the control of the existing hardware. To perform software installations and upgrades of worker nodes manually is a time consuming task especially if the number of worker nodes is more than 300. In order to cope with this kind of problem, an automatic installation tool is needed through which it will be possible to perform all necessary actions at once.

The GoeGrid Rocks Tools [84] provides a customised installation mechanism with additional software packages at install-time. Typically, the kickstart file [85] contains the OS and other necessary packages, for example acrobat reader, vidyo connection, etc. Besides this, it provides the functionality of the automatic DNS server configuration. For example, if there is a new worker node, the details about it should be entered in the Rocks database. After that, the new DNS entry will be generated without requiring any changes in the DNS configuration file. Once the new worker node is successfully installed it will be attached to the monitoring for future system administration and maintenance. In the case of GoeGrid, the new installed worker nodes are attached to the Ganglia [86] monitoring tool.

Through the Rocks Tools the management and the control of existing hardware is easily adaptable and efficient.



Current Network Status
Last Updated: Wed Apr 27 16:34:46 CEST 2016
Updated every 90 seconds
Nagios® Core™ 3.2.3 - www.nagios.org
Logged in as *nagiosadmin*

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
26	3	0	0

All Problems	All Types
3	29

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
227	10	0	54	0

All Problems	All Types
64	291

Service Status Details For All Hosts

Host	Service	Status	Last Check	Duration	Attempt	Status Information
apel	Current Load	OK	04-27-2016 15:55:47	65d 19h 38m 59s	1/2	OK - load average: 0.00, 0.00, 0.00
	PING	OK	04-27-2016 16:29:00	65d 3h 5m 48s	1/15	PING OK - Packet loss = 0%, RTA = 0.45 ms
	Processes	OK	04-27-2016 15:50:04	65d 19h 44m 42s	1/2	PROCS OK: 91 processes with STATE = RSZDT
	Root Partition	OK	04-27-2016 15:53:10	65d 19h 41m 36s	1/2	DISK OK - free space: / 60851 MB (84% inode=98%):
	Swap Usage	OK	04-27-2016 15:56:15	65d 19h 38m 31s	1/2	SWAP OK - 100% free (4094 MB out of 4095 MB)
	var Partition	CRITICAL	04-27-2016 15:59:21	431d 16h 25m 26s	2/2	NRPE: Command 'check_var' not defined
bdii	Current Load	OK	04-27-2016 16:02:27	216d 16h 0m 14s	1/2	OK - load average: 0.02, 0.01, 0.00
	PING	OK	04-27-2016 16:25:33	216d 16h 57m 8s	1/15	PING OK - Packet loss = 0%, RTA = 0.28 ms
	Processes	OK	04-27-2016 16:08:38	216d 16h 54m 3s	1/2	PROCS OK: 177 processes with STATE = RSZDT
	Root Partition	OK	04-27-2016 16:11:44	216d 16h 50m 57s	1/2	DISK OK - free space: / 1921 MB (50% inode=78%):
	Swap Usage	OK	04-27-2016 15:43:59	65d 19h 50m 47s	1/2	SWAP OK - 100% free (511 MB out of 511 MB)
rocks6	Current Load	OK	04-27-2016 16:21:05	8d 0h 22m 19s	1/2	OK - load average: 0.02, 0.02, 0.05
	Current Users	OK	04-27-2016 16:39:10	1d 22h 4m 14s	1/2	USERS OK - 13 users currently logged in
	Processes	WARNING	04-27-2016 16:12:16	8d 1h 31m 8s	2/2	PROCS WARNING: 347 processes with STATE = RSZDT
	Root Partition	OK	04-27-2016 15:44:31	216d 18h 26m 48s	1/2	DISK OK - free space: / 998 MB (53% inode=87%):
	Swap Usage	OK	04-27-2016 15:47:37	627d 16h 0m 54s	1/2	SWAP OK - 96% free (982 MB out of 1023 MB)
se	Check Certificates	OK	04-27-2016 15:50:42	9d 6h 52m 42s	1/2	ALL CERTIFICATES ARE UP TO DATE
	Current Load	OK	04-27-2016 16:08:48	9d 6h 34m 36s	1/2	OK - load average: 0.52, 0.56, 0.67
	Current Users	OK	04-27-2016 15:56:55	9d 6h 46m 29s	1/2	USERS OK - 0 users currently logged in
	Disk Load	OK	04-27-2016 16:30:00	9d 6h 13m 24s	1/2	DISK OK - free space: /usr 2474 MB (64% inode=85%): /opt 130554 MB (52% inode=97%): / 888 MB (46% inode=88%): /tmp 1745 MB (91% inode=99%):
	Hardware RAID	OK	04-27-2016 16:03:05	9d 6h 40m 19s	1/2	Okay (OKY) RAID1 Optimal (OPT) Optimal (OPT) Ready (RDY) Standby (SBY)
	PING	OK	04-27-2016 16:36:11	9d 6h 47m 13s	1/15	PING OK - Packet loss = 0%, RTA = 0.23 ms
	Pbs Ping	OK	04-27-2016 16:09:17	9d 6h 34m 7s	1/2	PING OK - Packet loss = 0%, RTA = 0.48 ms
	Processes	OK	04-27-2016 16:12:23	9d 6h 31m 1s	1/2	PROCS OK: 275 processes with STATE = RSZDT
	Root Partition	OK	04-27-2016 15:44:37	9d 5h 58m 47s	1/2	DISK OK - free space: / 888 MB (46% inode=88%):
	Software RAID Check	OK	04-27-2016 16:33:43	9d 6h 49m 41s	1/3	OK: md:md0:UUUUUUUUUUUUUUU
	Swap Usage	OK	04-27-2016 15:50:49	9d 6h 52m 35s	1/2	SWAP OK - 47% free (960 MB out of 2047 MB)
	dcachestatus	OK	04-27-2016 15:53:54	9d 6h 49m 30s	1/2	dCache OK : DOMAIN->STATUS dCacheDomain->running adminDoorDomain->running httpdDomain->running utilityDomain->running gPlazma-se-goegridDomain->running infoDomain->running statisticsDomain->running srm-se-goegridDomain->running
	var Partition	OK	04-27-2016 15:57:00	9d 6h 46m 24s	1/2	DISK OK - free space: /var 6057 MB (79% inode=87%):

Figure 3.13: Nagios GoeGrid web interface.

Another management tool, which is used in GoeGrid is CFengine [87]. CFengine is widly used for the large scale computing facilities. It is a language-based system administration tool where system maintenance is automated and the worker nodes configurations are defined in a central file.

GoeGrid uses CFengine in order to automate a large number of routine administrative tasks, which are requiring manpower.

Middleware

For the HEP experiments, the requirements are made by the WLCG. The main requirements are the choice of the grid middleware, which is gLite and the operating system for the WNs, which is Scientific Linux CERN 6 (SLC6) [88]. SL6 is compatible with all community specific requirements, the applications and the middleware.

Chapter 4

ATLAS Computing

4.1 ATLAS Computing Model

The ATLAS computing model [89] is mainly designed to allow all ATLAS members to have quick access to all reconstructed data for running different analysis. Additionally, it provides access to the corresponding raw data for managing calibration, monitoring and other activities.

This model is using the Grid Computing concept (see Section 2.1) and is based on the grid system itself. Therefore, there are a number of different software that are developed and designed by CERN for fulfilling the ATLAS requirements. The developed software have different purposes such as monitoring, job submission, data transfer and others. This chapter will describe the main software that is used in ATLAS, such as AGIS (see Section 4.2), PanDA (see Section 4.3), DDM (see Section 4.4), SSB (see Section 4.5) and others.

4.2 ATLAS Grid Information System (AGIS)

Each year the ATLAS experiment produces petabytes of data that needs to be distributed all over the world according to the ATLAS computing model. It provides quick access to all the necessary data to all ATLAS Collaboration members, such as reconstructed data, raw data for calibration and alignment activities, according to their roles or rights in the community.

The ATLAS Computing Infrastructure proposed the creation or development of a central information system to store various parameters and configurations that are required by many other ATLAS software components. A centralised information system solves the problem of data inconsistencies between data stored in various ATLAS services and the data stored in various external databases.

The ATLAS Grid Information System (AGIS) [90] is the central information system that integrates different configuration parameters for operating the ATLAS Distributed Computing (ADC) [91] system and its services. This includes the collection of different parameters from independent sources, such as the gLite BDII (see Section 3.2.2), the Grid Operations Centre Database (GOCDB) [92] and the Open Science Grid Information services (MyOSG , OSG Information Management System - OIM) [93]. Currently, AGIS is the main source of information for the ADC applications, the ATLAS Distributed Data Management (DDM) system (see Section 4.4) and the ATLAS Production and Distributed Analysis (PanDA) workload management system (see Section 4.3).

The architecture of AGIS is based on the client-server computing model. It is written in Python and uses the Django [94] framework. As a databases backend, it uses the Oracle database. AGIS caches information from different sources. Data synchronisation in AGIS is done by different agents who periodically check the sources via standard interfaces and in case of changes update the content of the AGIS database. The AGIS system provides different APIs, WebUIs, CLIs for data retrieval and management.

AGIS stores the complete information about the ATLAS topology such as clouds, regional centers, pledges, sites specifics (geography, time zone) and other information. It also stores service and resource information of different ATLAS Tier sites, such as the description of the Local File Catalog (LFC) [95], the File Transfer Service [71], [96], the Computing Element (CE) (see Section 2.1.4), the Storage Resource Manager [97], Squid [98], Frontier [99], the perfSONAR [100] services and others.

The web interface provides users the access to the AGIS information through a web browser. Users can visualise and modify the stored information depending on their roles and privileges. To update the information, users need to be granted a corresponding *write* permission. AGIS uses X.509 user certificates for authentication and authorisation.

AGIS shows the information for all the ATLAS sites.

Figure 4.1 shows an example of the GoeGrid services displayed in the AGIS web interface.

The table shows all running service types in the GoeGrid with corresponding endpoints and statuses.

Figure 4.2 shows an example of the GoeGrid PanDA queues displayed in the AGIS web interface.

The table shows PanDA analy, production and mcore queues with corresponding statuses for the GoeGrid site.

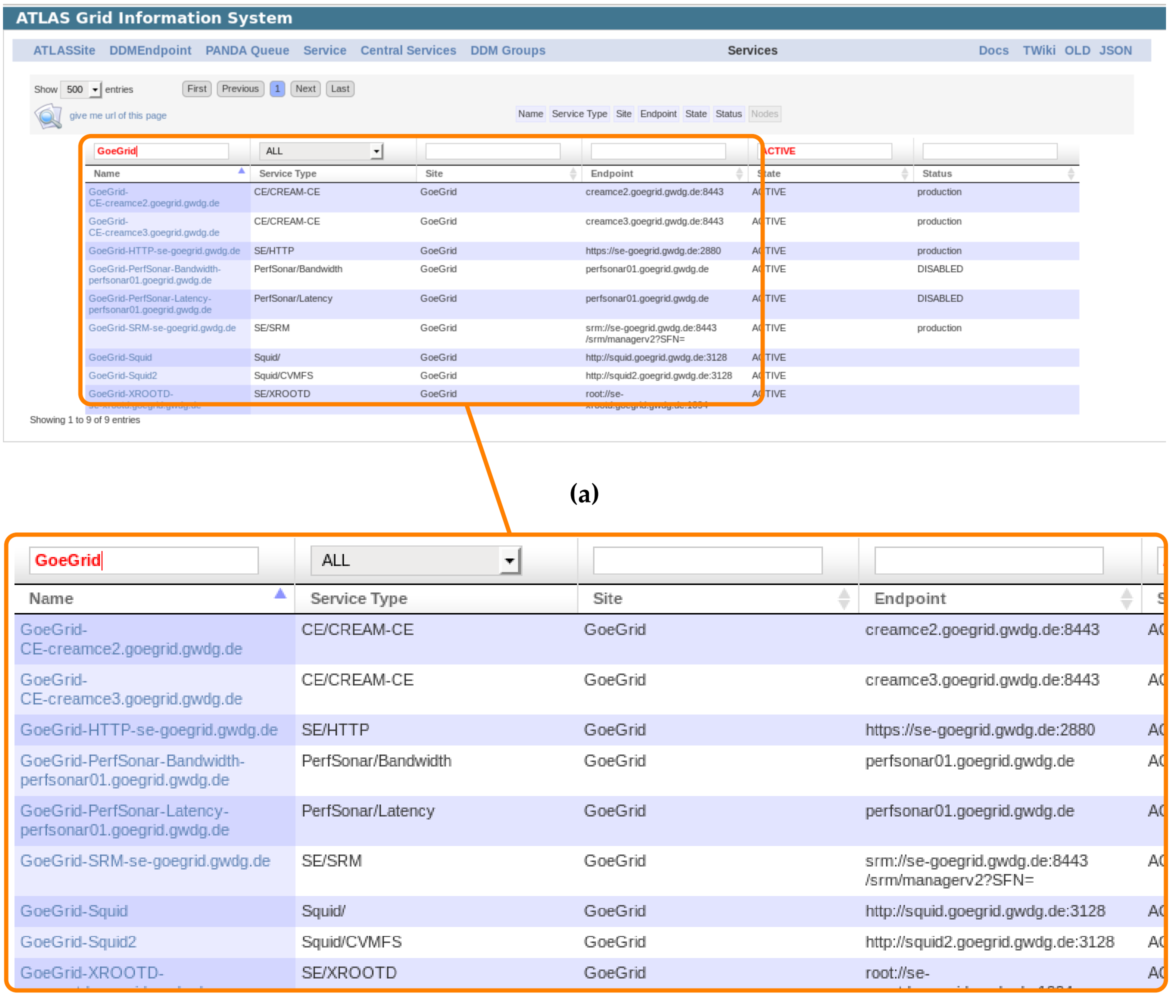
The AGIS system is currently in production and represents the main source of information for all the ADC applications and services.

4.3 Distributed Job Management System

The Production and Distributed Analysis (PanDA) [103] system was first developed by ATLAS collaboration members in 2005. Initially it was developed for US based ATLAS production and analysis jobs and later in 2007, it was adapted to the needs of the whole ATLAS collaboration.

PanDA is a workload management system for processing distributed analysis and production jobs. PanDA plays a key role in the ATLAS distributed computing infrastructure. It is the only system that processes all ATLAS Monte-Carlo simulation and data reprocessing jobs. Each day PanDA processes approximately a million jobs. Users submit their job to the PanDA system via a Python/HTTP client interface. This interface is used for analysis and production jobs.

The main component of the PanDA system is the PanDA server. The PanDA server takes care of task queues and centrally manages job execution. Submitted user jobs go to the task queue where a brokerage module operates them according to their input data, type, priority, available WN resources and other parameters. Then pilots [104]



ATLAS Grid Information System

ATLASSite DDMEndpoint PANDA Queue Service Central Services DDM Groups
Services
Docs TWiki OLD JSON
Show 500 entries
First Previous 1 Next Last
give me url of this page
Name Service Type Site Endpoint State Status Nodes

GoeGrid	ALL			ACTIVE	
Name	**Service Type**	**Site**	**Endpoint**	**State**	**Status**
GoeGrid-CE-creamce2.goegrid.gwdg.de	CE/CREAM-CE	GoeGrid	creamce2.goegrid.gwdg.de:8443	ACTIVE	production
GoeGrid-CE-creamce3.goegrid.gwdg.de	CE/CREAM-CE	GoeGrid	creamce3.goegrid.gwdg.de:8443	ACTIVE	production
GoeGrid-HTTP-se-goegrid.gwdg.de	SE/HTTP	GoeGrid	https://se-goegrid.gwdg.de:2880	ACTIVE	production
GoeGrid-PerfSonar-Bandwidth-perfsonar01.goegrid.gwdg.de	PerfSonar/Bandwidth	GoeGrid	perfsonar01.goegrid.gwdg.de	ACTIVE	DISABLED
GoeGrid-PerfSonar-Latency-perfsonar01.goegrid.gwdg.de	PerfSonar/Latency	GoeGrid	perfsonar01.goegrid.gwdg.de	ACTIVE	DISABLED
GoeGrid-SRM-se-goegrid.gwdg.de	SE/SRM	GoeGrid	srm://se-goegrid.gwdg.de:8443 /srm/managerv2?SFN=	ACTIVE	production
GoeGrid-Squid	Squid/	GoeGrid	http://squid.goegrid.gwdg.de:3128	ACTIVE	
GoeGrid-Squid2	Squid/CVMFS	GoeGrid	http://squid2.goegrid.gwdg.de:3128	ACTIVE	
GoeGrid-XROOTD-	SE/XROOTD	GoeGrid	root://se-	ACTIVE	

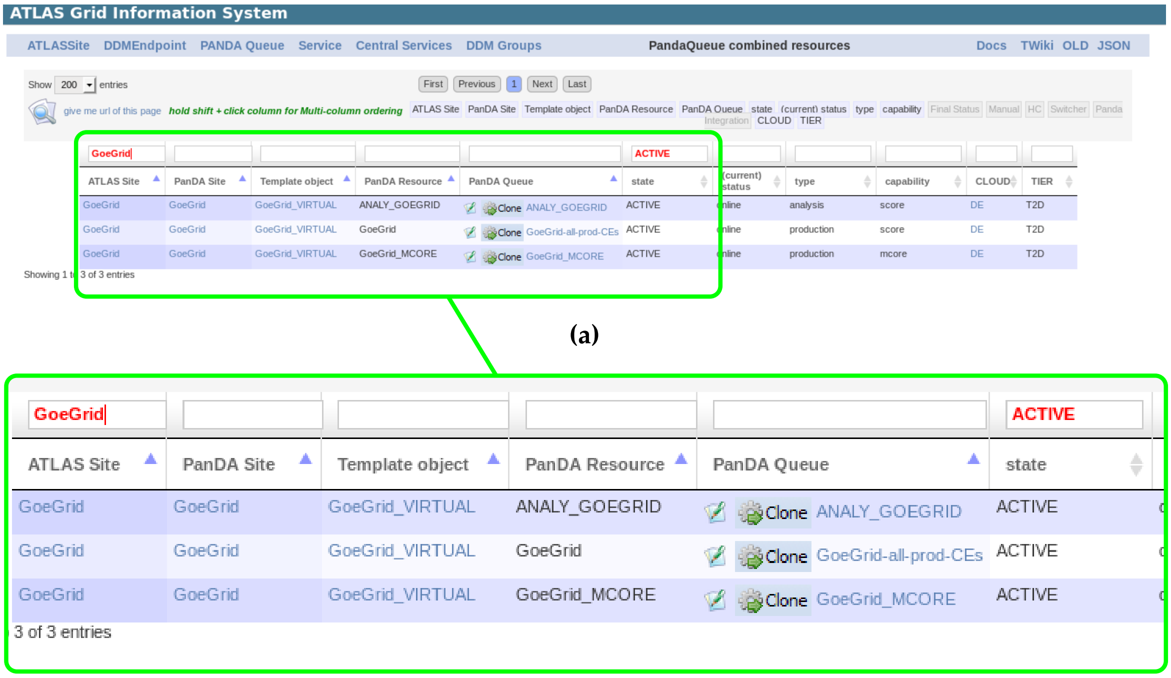
Showing 1 to 9 of 9 entries

(a)

GoeGrid	ALL		
Name	**Service Type**	**Site**	**Endpoint**
GoeGrid-CE-creamce2.goegrid.gwdg.de	CE/CREAM-CE	GoeGrid	creamce2.goegrid.gwdg.de:8443
GoeGrid-CE-creamce3.goegrid.gwdg.de	CE/CREAM-CE	GoeGrid	creamce3.goegrid.gwdg.de:8443
GoeGrid-HTTP-se-goegrid.gwdg.de	SE/HTTP	GoeGrid	https://se-goegrid.gwdg.de:2880
GoeGrid-PerfSonar-Bandwidth-perfsonar01.goegrid.gwdg.de	PerfSonar/Bandwidth	GoeGrid	perfsonar01.goegrid.gwdg.de
GoeGrid-PerfSonar-Latency-perfsonar01.goegrid.gwdg.de	PerfSonar/Latency	GoeGrid	perfsonar01.goegrid.gwdg.de
GoeGrid-SRM-se-goegrid.gwdg.de	SE/SRM	GoeGrid	srm://se-goegrid.gwdg.de:8443 /srm/managerv2?SFN=
GoeGrid-Squid	Squid/	GoeGrid	http://squid.goegrid.gwdg.de:3128
GoeGrid-Squid2	Squid/CVMFS	GoeGrid	http://squid2.goegrid.gwdg.de:3128
GoeGrid-XROOTD-	SE/XROOTD	GoeGrid	root://se-

(b)

Figure 4.1: GoeGrid services displayed in AGIS web interface [101].



ATLAS Grid Information System

ATLASSite DDMEndpoint PANDA Queue Service Central Services DDM Groups PandaQueue combined resources Docs TWiki OLD JSON

Show 200 entries First Previous 1 Next Last

give me url of this page *hold shift + click column for Multi-column ordering* ATLAS Site PanDA Site Template object PanDA Resource PanDA Queue state (current) status type capability Final Status Manual HC Switcher Panda Integration CLOUD TIER

ATLAS Site	PanDA Site	Template object	PanDA Resource	PanDA Queue	state	(current) status	type	capability	CLOUD	TIER
GoeGrid					ACTIVE					
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	ANALY_GOEGRID	Clone ANALY_GOEGRID	ACTIVE	online	analysis	score	DE	T2D
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	GoeGrid	Clone GoeGrid-all-prod-CEs	ACTIVE	online	production	score	DE	T2D
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	GoeGrid_MCORE	Clone GoeGrid_MCORE	ACTIVE	online	production	mcore	DE	T2D

Showing 1 to 3 of 3 entries

(a)

ATLAS Site	PanDA Site	Template object	PanDA Resource	PanDA Queue	state
GoeGrid					ACTIVE
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	ANALY_GOEGRID	Clone ANALY_GOEGRID	ACTIVE
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	GoeGrid	Clone GoeGrid-all-prod-CEs	ACTIVE
GoeGrid	GoeGrid	GoeGrid_VIRTUAL	GoeGrid_MCORE	Clone GoeGrid_MCORE	ACTIVE

3 of 3 entries

(b)

Figure 4.2: GoeGrid PanDA queues displayed in AGIS web interface [102].

retrieve jobs from the task queue and run them into the available WNs. Pilots use resources efficiently. If there are no available jobs they exit immediately and the submission rate is regulated according to the load. Pilots execute jobs, detect stuck processes, recover failed jobs and report the job status to the PanDA server. The schematic view of the PanDA system is shown in Figure 4.3.

PanDA monitor is the web interface to the PanDA system. It provides summaries of processing per cloud/site, logfiles and shows the status of tasks/jobs. The table (Figure 4.4) displays job information for the GoeGrid site.

The most important parameters of the table are marked in red. The *computingsite* parameter shows the number of running jobs in GoeGrid, which is in this example 2568. The *jeditaskid* parameter lists the tasks (job set) and the number of jobs per task. In this example there are 71 running tasks and 2568 running jobs in total. The *jobstatus* parameter shows the status of jobs. In this example the number of failed jobs is 1424. By clicking on the number it is possible to see more detailed information about the failed jobs. Detailed information is written in logfiles. The *processingtype* parameter lists the task types running on the system.

The BigPanDA project was established in September 2012 as an extension of the PanDA project. The BigPanDA project is more generalised in terms of usage by other experiments. It extended its availability for High Performance Computing (HPC) platforms and has intelligent network awareness.

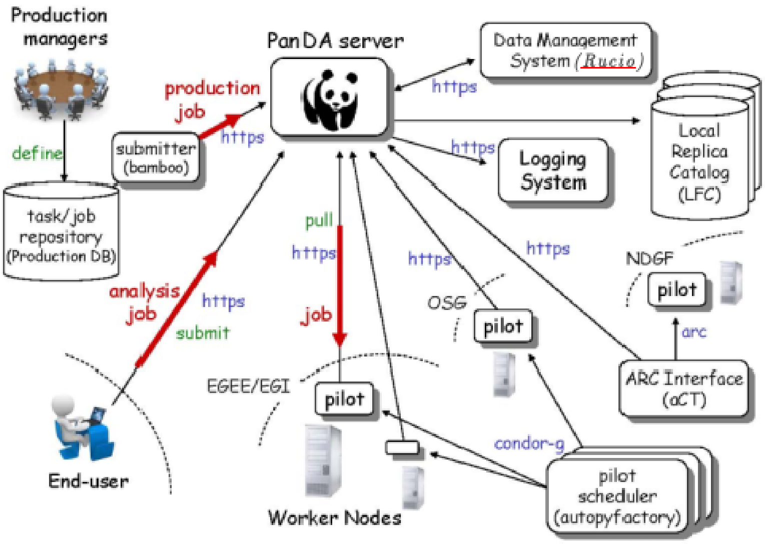
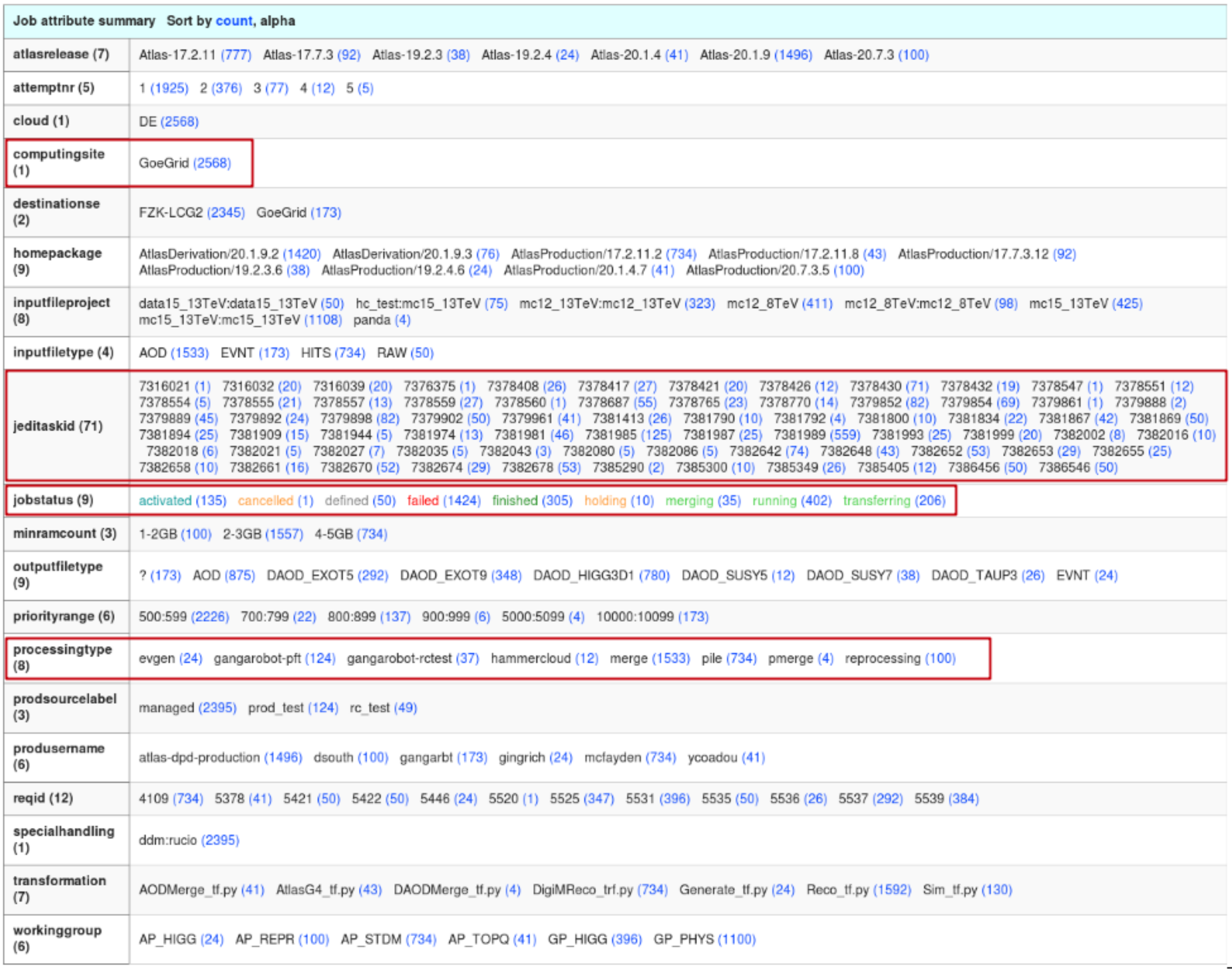


Figure 4.3: Schematic view of the PanDA system. Taken from [103].



Job attribute summary Sort by count, alpha	
atlasrelease (7)	Atlas-17.2.11 (777) Atlas-17.7.3 (92) Atlas-19.2.3 (38) Atlas-19.2.4 (24) Atlas-20.1.4 (41) Atlas-20.1.9 (1496) Atlas-20.7.3 (100)
attemptnr (5)	1 (1925) 2 (376) 3 (77) 4 (12) 5 (5)
cloud (1)	DE (2568)
computingsite (1)	GoeGrid (2568)
destinationse (2)	FZK-LCG2 (2345) GoeGrid (173)
homepackage (9)	AtlasDerivation/20.1.9.2 (1420) AtlasDerivation/20.1.9.3 (76) AtlasProduction/17.2.11.2 (734) AtlasProduction/17.2.11.8 (43) AtlasProduction/17.7.3.12 (92) AtlasProduction/19.2.3.6 (38) AtlasProduction/19.2.4.6 (24) AtlasProduction/20.1.4.7 (41) AtlasProduction/20.7.3.5 (100)
inputfileproject (8)	data15_13TeV:data15_13TeV (50) hc_test:mc15_13TeV (75) mc12_13TeV:mc12_13TeV (323) mc12_8TeV (411) mc12_8TeV:mc12_8TeV (98) mc15_13TeV (425) mc15_13TeV:mc15_13TeV (1108) panda (4)
inputfiletype (4)	AOD (1533) EVNT (173) HITS (734) RAW (50)
jeditaskid (71)	7316021 (1) 7316032 (20) 7316039 (20) 7376375 (1) 7378408 (26) 7378417 (27) 7378421 (20) 7378426 (12) 7378430 (71) 7378432 (19) 7378547 (1) 7378551 (12) 7378554 (5) 7378555 (21) 7378557 (13) 7378559 (27) 7378560 (1) 7378687 (55) 7378765 (23) 7378770 (14) 7379852 (82) 7379854 (69) 7379861 (1) 7379888 (2) 7379889 (45) 7379892 (24) 7379898 (82) 7379902 (50) 7379961 (41) 7381413 (26) 7381790 (10) 7381792 (4) 7381800 (10) 7381834 (22) 7381867 (42) 7381869 (50) 7381894 (25) 7381909 (15) 7381944 (5) 7381974 (13) 7381981 (46) 7381985 (125) 7381987 (25) 7381989 (559) 7381993 (25) 7381999 (20) 7382002 (8) 7382016 (10) 7382018 (6) 7382021 (5) 7382027 (7) 7382035 (5) 7382043 (3) 7382080 (5) 7382086 (5) 7382642 (74) 7382648 (43) 7382652 (53) 7382653 (29) 7382655 (25) 7382658 (10) 7382661 (16) 7382670 (52) 7382674 (29) 7382678 (53) 7385290 (2) 7385300 (10) 7385349 (26) 7385405 (12) 7386456 (50) 7386546 (50)
jobstatus (9)	activated (135) cancelled (1) defined (50) failed (1424) finished (305) holding (10) merging (35) running (402) transferring (206)
minramcount (3)	1-2GB (100) 2-3GB (1557) 4-5GB (734)
outputfiletype (9)	? (173) AOD (875) DAOD_EXOT5 (292) DAOD_EXOT9 (348) DAOD_HIGG3D1 (780) DAOD_SUSY5 (12) DAOD_SUSY7 (38) DAOD_TAUP3 (26) EVNT (24)
priorityrange (6)	500:599 (2226) 700:799 (22) 800:899 (137) 900:999 (6) 5000:5099 (4) 10000:10099 (173)
processingtype (8)	evgen (24) gangarobot-pft (124) gangarobot-rctest (37) hammercloud (12) merge (1533) pile (734) pmerge (4) reprocessing (100)
prodsourcelabel (3)	managed (2395) prod_test (124) rc_test (49)
produsername (6)	atlas-dpd-production (1496) dsouth (100) gangarbt (173) gingrich (24) mcfayden (734) ycoadou (41)
reqid (12)	4109 (734) 5378 (41) 5421 (50) 5422 (50) 5446 (24) 5520 (1) 5525 (347) 5531 (396) 5535 (50) 5536 (26) 5537 (292) 5539 (384)
specialhandling (1)	ddm:rucio (2395)
transformation (7)	AODMerge_tf.py (41) AtlasG4_tf.py (43) DAODMerge_tf.py (4) DigiMReco_trf.py (734) Generate_tf.py (24) Reco_tf.py (1592) Sim_tf.py (130)
workinggroup (6)	AP_HIGG (24) AP_REPR (100) AP_STDM (734) AP_TOPQ (41) GP_HIGG (396) GP_PHYS (1100)

Figure 4.4: PanDA monitor GoeGrid example [105].

4.4 Distributed Data Management System

Each year, the ATLAS experiment produces petabytes of data that need to be distributed and stored in the Worldwide LHC Computing Grid for future processing and analysis. The management of these data is done by the ATLAS Distributed Data Management system (DDM), called Rucio [106]. Rucio is meant to manage the large volumes of data.

The DDM project was established in spring 2005 to develop the system called Don Quijote 2 (DQ2). The main idea was to create a system that is scalable, robust, flexible and will fulfil the needs of the ATLAS computing model. This means the creation of a complete data flow of the experiment which includes everything: starting from archiving of raw data until individual physics analysis at home institutes. DQ2 has evolved rapidly to help ATLAS Computing operations to manage large volumes of data across many grid sites at which ATLAS runs, and to help ATLAS physicists get access to these data. However, DQ2 reached its limits in terms of scalability, requiring a large number of support staff to operate and it was hard to keep up with new technologies. DQ2 was widely used between 2008-2014. The replacement of DQ2 is the new project called Rucio. Rucio is more flexible and solves almost all open problems of DQ2. Rucio is meant to cope with the requirements of LHC Run 2 in terms of scalability and the expected huge amount of data. Since January 2015, DQ2 was officially decommissioned and Rucio started to operate as the new DDM system.

Rucio accounts

The entry point to the Rucio system is a Rucio account. A Rucio account represents individual users or groups of users or any service, which needs to perform activities in the ATLAS collaboration. Rucio users identify themselves by an X.509 certificate. User credentials can map to one or more accounts (N:M mapping). Rucio checks the permissions of the credentials to allow the use of a Rucia account. Figure 4.5 displays an example of the mapping between user credentials and Rucio accounts.

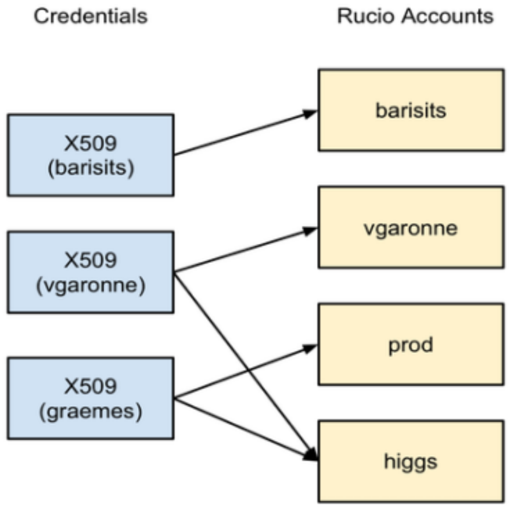


Figure 4.5: Mapping between user credentials and Rucio accounts. Taken from [107].

Rucio data identifiers, scope and metadata attributes

ATLAS data is physically stored in files. The data itself consists of persistent C++ objects associated with physics events. In Rucio, the smallest operational unit of data are files. Files can be grouped into datasets (set of multiple files). Datasets from their side can be grouped into containers (set of datasets). Files, datasets and containers (Figure 4.6) follow an identical naming scheme which consists of a name and the scope. The combination of both is called data identifier (DI).

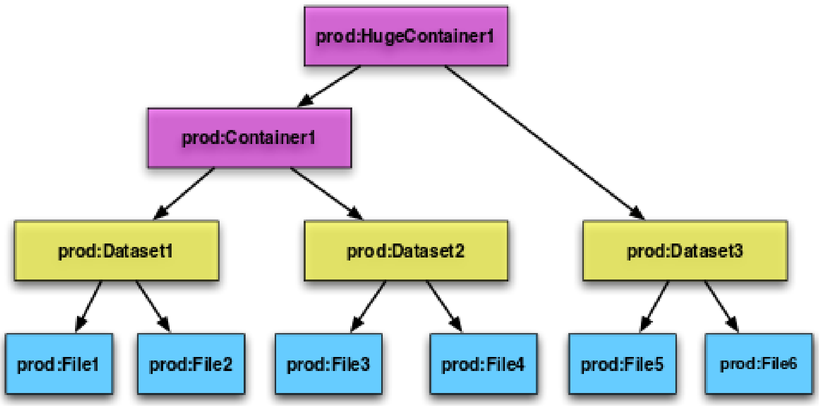


Figure 4.6: Aggregation hierarchy example. Taken from [106].

Rucio account holders have read access to all scopes and write access to their own scope. Privileged accounts have write access to multiple scopes. Files, datasets and containers are uniquely identified in the system. Even if data are removed from the system the DI cannot be reused.

Rucio Storage Element

A Rucio Storage Element (RSE) is a repository for physical files. It is the smallest unit of storage space addressable within Rucio. The RSE has a unique identifier and a set of meta attributes such as supported protocols, storage type, physical space properties, geographical zone, etc.

RSEs can be grouped in many logical ways. For example the DE RSEs and the Tier-1 RSEs. It is possible to refer to a group of RSEs using metadata attributes or by simple calculation of RSEs.

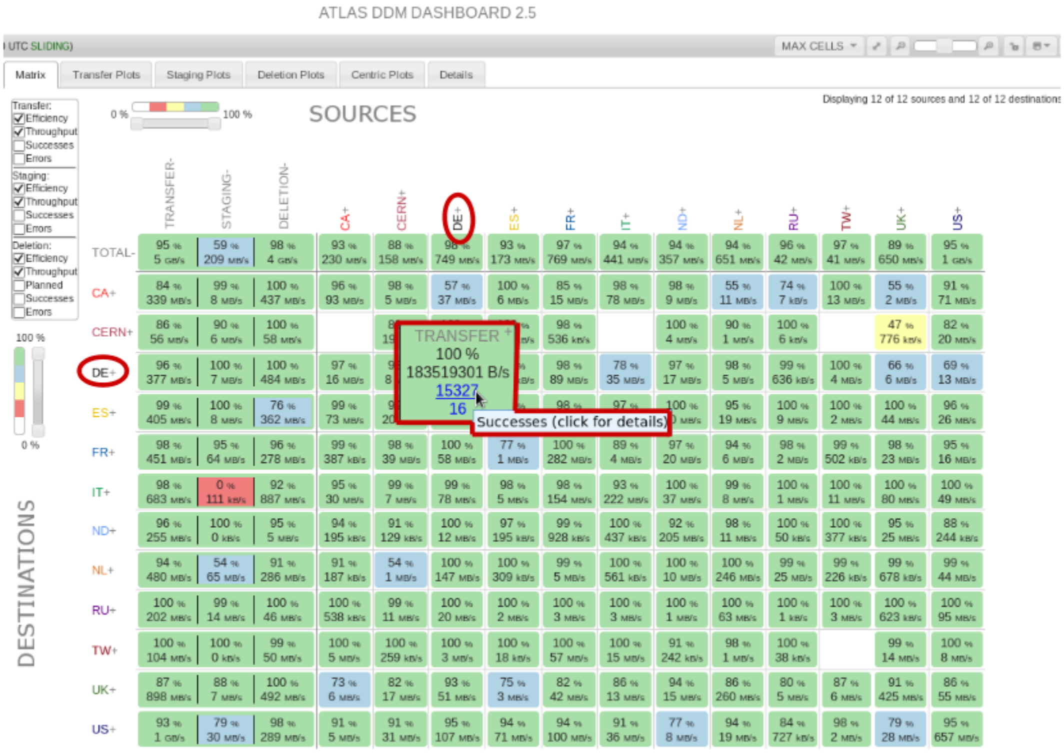
In Rucio, caching is a mutable service. The cache is usually managed by an external process or application, such as the PanDA system.

Rucio architecture

The Rucio system is based on a distributed architecture. It provides a command line, python clients and APIs for users. After successfully passing through the authentication and authorisation component, the user gets a short lifetime token. Once the user is authenticated the RSE provides the token for the next interaction with the system. The RSE component cares for the interactions within various grid middleware tools and storage systems. The Rucio daemon is one of the main components that guides the whole system. For example, the Conveyor of the Rucio daemon takes care of the file transfers and the Reaper deals with file replica deletion. In terms of database usage, Rucio supports several relational database management systems (RDBMS) like Oracle, PostgreSQL or MySQL. The Rucio Probes component takes care of performing various tests. It checks interactions within Rucio and other ATLAS software such as the ATLAS Meta-data interface (AMI) [109] and AGIS. The Rucio Analytics component is responsible for accounting the data and making up reports.

ATLAS Distributed Data Management (DDM) dashboard

The DDM dashboard [110] is a web interface while it is used to maintain the high reliability of the DDM (Rucio) system. It is a widely used interface. Through this interface it is possible to follow the transfers within the ATLAS clouds/sites (Figure 4.7). In this example transfers from the DE cloud to the DE cloud are at 100%. The number of successful transfers is 15327 and there are only 16 failed transfers. By navigating deeper, the detailed information about transferred datasets, such as time, name, etc., the reason of failed transfers can be seen.



Displaying 12 of 12 sources and 12 of 12 destinations

Transfer:
☑ Efficiency
☑ Throughput
□ Successes
□ Errors

Staging:
☑ Efficiency
☑ Throughput
□ Successes
□ Errors

Deletion:
☑ Efficiency
☑ Throughput
□ Planned
□ Successes
□ Errors

	TRANSFER-	STAGING-	DELETION-	CA+	CERN+	DE+	ES+	FR+	IT+	ND+	NL+	RU+	TW+	UK+	US+
TOTAL-	95 % 5 GB/s	59 % 209 MB/s	98 % 4 GB/s	93 % 230 MB/s	88 % 158 MB/s	98 % 749 MB/s	93 % 173 MB/s	97 % 769 MB/s	94 % 441 MB/s	94 % 357 MB/s	94 % 651 MB/s	96 % 42 MB/s	97 % 41 MB/s	89 % 650 MB/s	95 % 1 GB/s
CA+	84 % 339 MB/s	99 % 8 MB/s	100 % 437 MB/s	96 % 93 MB/s	98 % 5 MB/s	57 % 37 MB/s	100 % 6 MB/s	85 % 15 MB/s	98 % 78 MB/s	98 % 9 MB/s	55 % 11 MB/s	74 % 7 kB/s	100 % 13 MB/s	55 % 2 MB/s	91 % 71 MB/s
CERN+	86 % 56 MB/s	90 % 6 MB/s	100 % 58 MB/s		8[illegible] 19[illegible]	TRANSFER + 100 % 183519301 B/s 15327 16 Successes (click for details)	[illegible]	98 % 536 kB/s		100 % 4 MB/s	90 % 1 MB/s	100 % 6 kB/s		47 % 776 kB/s	82 % 20 MB/s
DE+	96 % 377 MB/s	100 % 7 MB/s	100 % 484 MB/s	97 % 16 MB/s	9[illegible] 8[illegible]		[illegible] kB/s	98 % 89 MB/s	78 % 35 MB/s	97 % 17 MB/s	98 % 5 MB/s	99 % 636 kB/s	100 % 4 MB/s	66 % 6 MB/s	69 % 13 MB/s
ES+	99 % 405 MB/s	100 % 8 MB/s	76 % 362 MB/s	99 % 73 MB/s	9[illegible] 20[illegible]		[illegible]	98 % [illegible]	97 % [illegible]	100 % [illegible] MB/s	95 % 19 MB/s	100 % 9 MB/s	100 % 2 MB/s	100 % 44 MB/s	96 % 26 MB/s
FR+	98 % 451 MB/s	95 % 64 MB/s	96 % 278 MB/s	99 % 387 kB/s	98 % 39 MB/s	100 % 58 MB/s	77 % 1 MB/s	100 % 282 MB/s	89 % 4 MB/s	97 % 20 MB/s	94 % 6 MB/s	98 % 2 MB/s	99 % 502 kB/s	98 % 23 MB/s	95 % 16 MB/s
IT+	98 % 683 MB/s	0 % 111 kB/s	92 % 887 MB/s	95 % 30 MB/s	99 % 7 MB/s	99 % 78 MB/s	98 % 5 MB/s	98 % 154 MB/s	93 % 222 MB/s	100 % 37 MB/s	99 % 8 MB/s	100 % 1 MB/s	100 % 11 MB/s	100 % 80 MB/s	100 % 49 MB/s
ND+	96 % 255 MB/s	100 % 0 kB/s	95 % 5 MB/s	94 % 195 kB/s	91 % 129 kB/s	100 % 12 MB/s	97 % 195 kB/s	99 % 928 kB/s	100 % 437 kB/s	92 % 205 MB/s	98 % 11 MB/s	100 % 50 kB/s	100 % 377 kB/s	95 % 25 MB/s	88 % 244 kB/s
NL+	94 % 480 MB/s	54 % 65 MB/s	91 % 286 MB/s	91 % 187 kB/s	54 % 1 MB/s	100 % 147 MB/s	100 % 309 kB/s	99 % 5 MB/s	100 % 561 kB/s	100 % 10 MB/s	100 % 246 MB/s	99 % 25 MB/s	99 % 226 kB/s	99 % 678 kB/s	99 % 44 MB/s
RU+	100 % 202 MB/s	99 % 14 MB/s	100 % 46 MB/s	100 % 538 kB/s	99 % 11 MB/s	100 % 20 MB/s	100 % 2 MB/s	100 % 3 MB/s	100 % 3 MB/s	100 % 1 MB/s	100 % 63 MB/s	100 % 1 kB/s	100 % 3 MB/s	100 % 623 kB/s	100 % 95 MB/s
TW+	100 % 104 MB/s	100 % 0 kB/s	99 % 50 MB/s	100 % 5 MB/s	100 % 259 kB/s	100 % 3 MB/s	100 % 18 kB/s	100 % 57 MB/s	100 % 15 MB/s	91 % 242 kB/s	98 % 1 MB/s	100 % 38 kB/s		99 % 14 MB/s	100 % 8 MB/s
UK+	87 % 898 MB/s	88 % 7 MB/s	100 % 492 MB/s	73 % 6 MB/s	82 % 17 MB/s	93 % 51 MB/s	75 % 3 MB/s	82 % 42 MB/s	86 % 13 MB/s	94 % 15 MB/s	86 % 260 MB/s	80 % 5 MB/s	87 % 6 MB/s	91 % 425 MB/s	86 % 55 MB/s
US+	93 % 1 GB/s	79 % 30 MB/s	98 % 289 MB/s	91 % 5 MB/s	91 % 31 MB/s	95 % 107 MB/s	94 % 71 MB/s	94 % 100 MB/s	91 % 36 MB/s	77 % 8 MB/s	94 % 19 MB/s	84 % 727 kB/s	98 % 2 MB/s	79 % 28 MB/s	95 % 657 MB/s

Figure 4.7: DDM Dashboard interface [110].

There are also statistics plots (Figure 4.8) that display the successful and failed transfers for a given time period.

The plots represent the transfer efficiency, the volume of data and the succeeded and failed transfers for the last 10 min interval for different clouds. By navigating deeper, the same plots but for the sites in the clouds can be seen.

The DDM Dashboard is used constantly by people who are taking computing shifts [112] to investigate failures and notify sites in case of failures.

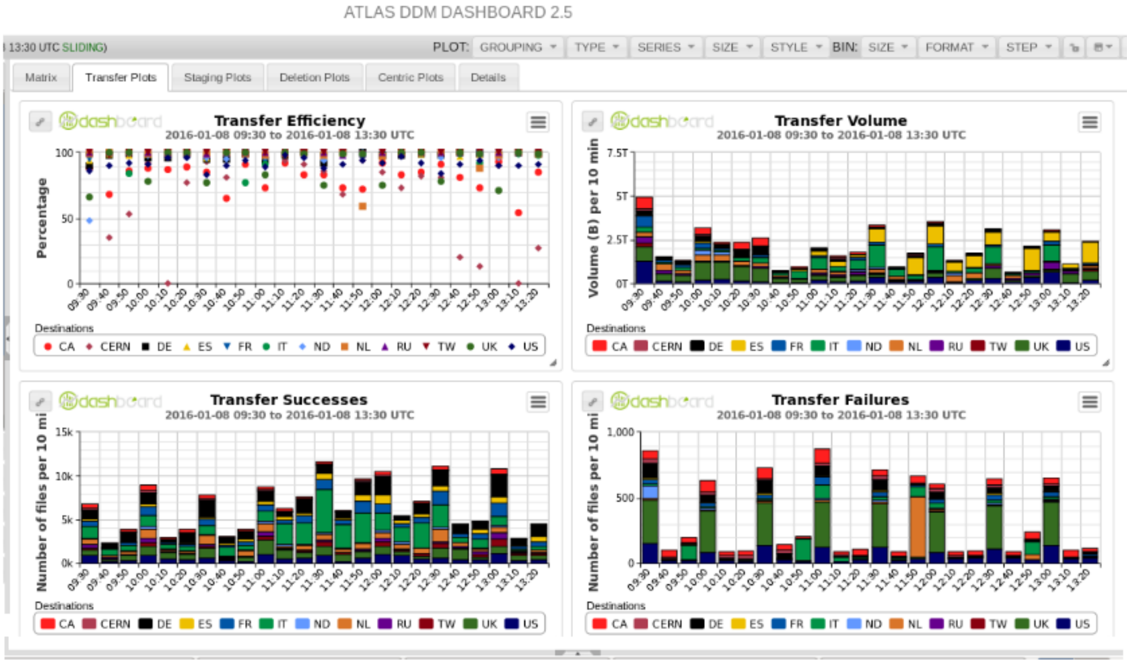

ATLAS DDM DASHBOARD 2.5
13:30 UTC SLIDING)
PLOT: GROUPING TYPE SERIES SIZE STYLE BIN: SIZE FORMAT STEP
Matrix
Transfer Plots
Staging Plots
Deletion Plots
Centric Plots
Details
Transfer Efficiency
2016-01-08 09:30 to 2016-01-08 13:30 UTC
Percentage
Transfer Volume
2016-01-08 09:30 to 2016-01-08 13:30 UTC
Volume (B) per 10 min
Transfer Successes
2016-01-08 09:30 to 2016-01-08 13:30 UTC
Number of files per 10 mi
Transfer Failures
2016-01-08 09:30 to 2016-01-08 13:30 UTC
Number of files per 10 mi
Destinations
CA CERN DE ES FR IT ND NL RU TW UK US

Figure 4.8: DDM Dashboard interface [111].

4.5 Site Status Board (SSB)

Taking into account the large amount of data and complexity of the whole infrastructure, it is a challenging task to provide a reliable and stable system. An effective monitoring and real-time action-taking is done constantly by ATLAS Distributed Computing (ADC) shifters and experts. Besides a manual monitoring and action-taking, an automated tool is needed to solve the problem of delay and to reduce the manual interactions. Based on monitoring data, the performance of different services can be measured and the problems can be detected easily. This is very important as the solution of existing problems can be done on time. Such a tool was designed and developed by ADC operations and is called the ATLAS Site Status Board (SSB) [113].

The main objective of the SSB is to collect, aggregate and display the monitoring information about the ADC critical activities on a single web interface. Beside this, the SSB includes the ATLAS Site Topology [114] and an automatic alert system. The SSB workflow is shown in Figure 4.9.

The collection and aggregation of information in the SSB is done by different scripts called Collectors. Collectors run periodically and collect data from various external monitoring resources. The collected information is stored in predefined data structures called metrics. Metrics typically contain quantitative or qualitative information about different services. They contain only current status information about the ATLAS site activities such as data analysis, data transfers and data processing. Old information is stored in a database to follow the status change of the ATLAS site activities. The SSB stores the historical information of the ATLAS site activities.

The SSB has a number of different web views. The SSB *default view* displays a huge table about all necessary information of the ATLAS sites. The SSB *shifter view* displays filtered information about all the ATLAS sites (Figure 4.10).

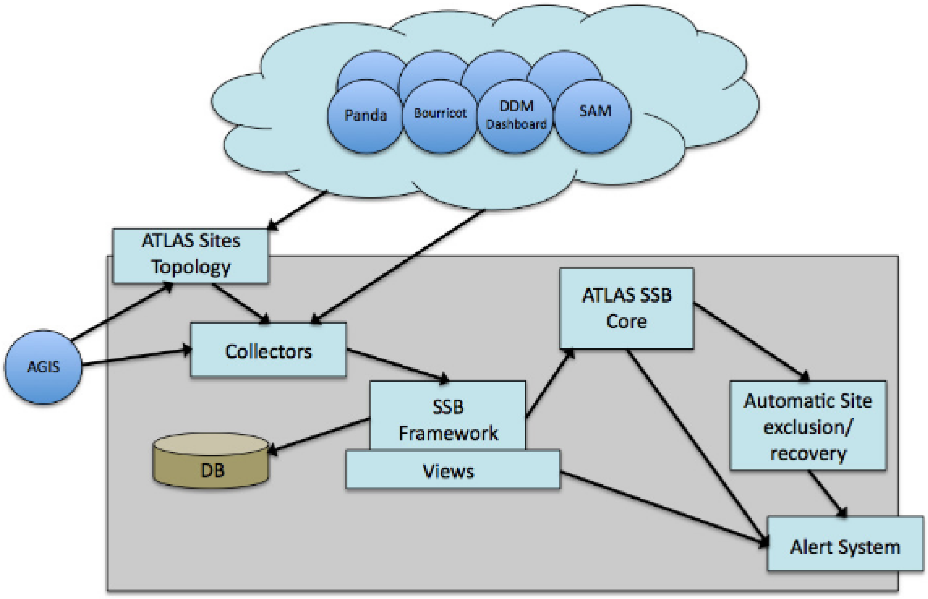


Figure 4.9: The workflow of the ATLAS SSB. Taken from [113].

By navigating deeper, it is possible to see detailed information from the source of the published values or statuses.

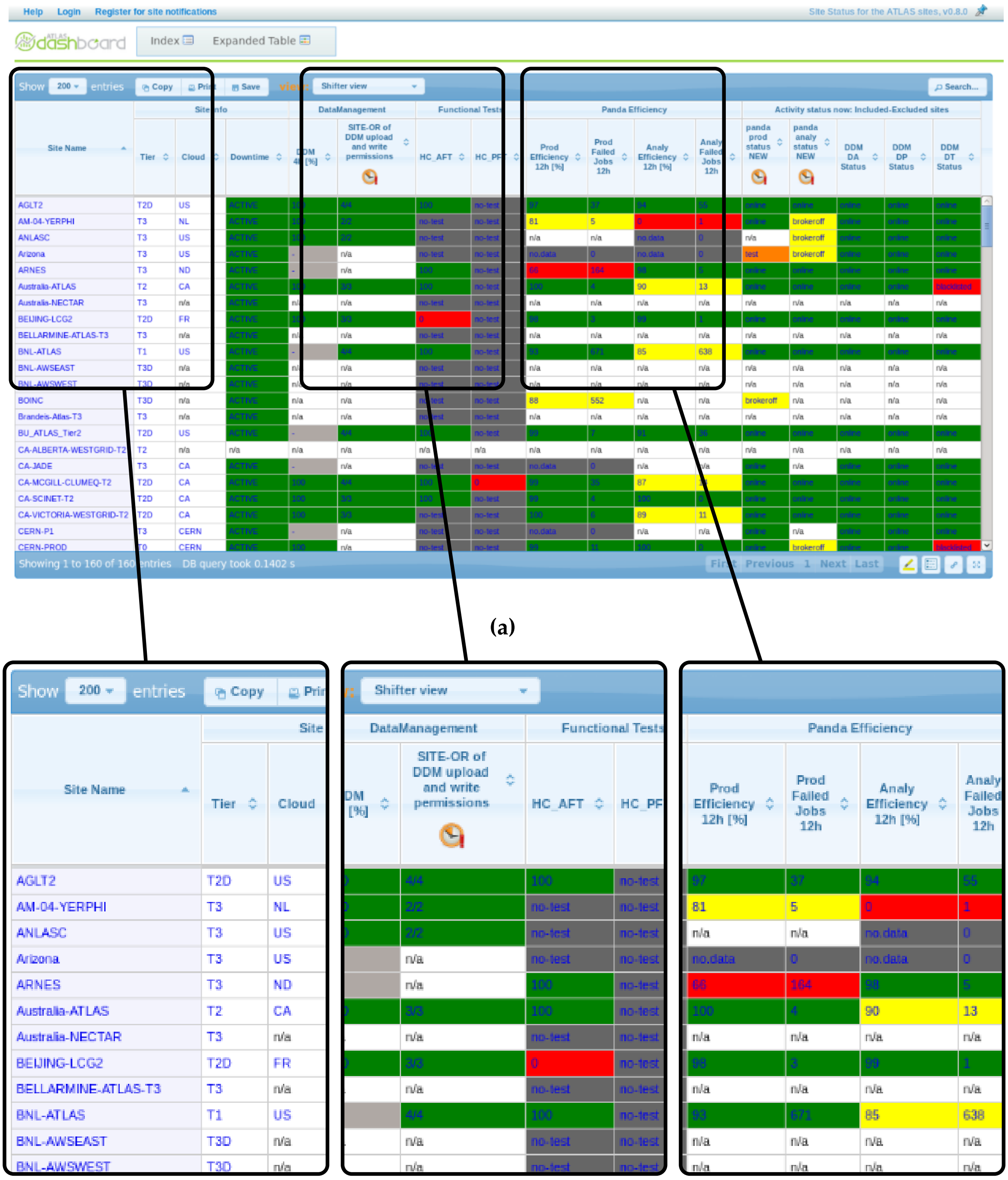
The SSB tool is a step towards achieving the main goal of ADC, to reduce the need for manual interaction in the ATLAS Grid Computing activities.

4.6 ATLAS Offline Software and Computing

The ATLAS offline software is meant to process the ATLAS trigger [116] and data acquisition [117] events, as well as to deliver the processed results to physicists within the ATLAS Collaboration. It also provides various tools for physicists to analyse the processed information and to get the physics results. A physics requirement is the ability to reproduce the data of physics process based on the detector data. The data processing requires corresponding simulation tools.

The Athena framework [118] is a common ATLAS–LHCb project and is used by other experiments as well. The Athena framework is designed to handle a wide range of data-processing applications. The component-based architecture provides the flexibility of the Athena framework. It allows the usage of shared components and the development of experiment specific components. The Athena framework is used for simulation, reconstruction and analysis.

Another commonly used software package is ROOT [119]. ROOT is an open-source software written in C++. It was developed by CERN. ROOT is mainly designed for particle physics data analysis, where scientists produce and analyse enormous amounts of complex data. It includes several specific libraries of this field. ROOT is widely used by several experiments in high-energy physics. It also can be used for astronomy and data mining.



	Site info			DataManagement		Functional Tests		Panda Efficiency				Activity status now: Included-Excluded sites				
Site Name	Tier	Cloud	Downtime	DDM 4h [%]	SITE-OR of DDM upload and write permissions	HC_AFT	HC_PFT	Prod Efficiency 12h [%]	Prod Failed Jobs 12h	Analy Efficiency 12h [%]	Analy Failed Jobs 12h	panda prod status NEW	panda analy status NEW	DDM DA Status	DDM DP Status	DDM DT Status
AGLT2	T2D	US	ACTIVE	100	4/4	100	no-test	97	37	94	55	online	online	online	online	online
AM-04-YERPHI	T3	NL	ACTIVE	100	2/2	no-test	no-test	81	5	0	1	online	brokeroff	online	online	online
ANLASC	T3	US	ACTIVE	100	2/2	no-test	no-test	n/a	n/a	no.data	0	n/a	brokeroff	online	online	online
Arizona	T3	US	ACTIVE	-	n/a	no-test	no-test	no.data	0	no.data	0	test	brokeroff	online	online	online
ARNES	T3	ND	ACTIVE	-	n/a	100	no-test	66	164	98	5	online	online	online	online	online
Australia-ATLAS	T2	CA	ACTIVE	100	3/3	100	no-test	100	4	90	13	online	online	online	online	blacklisted
Australia-NECTAR	T3	n/a	ACTIVE	n/a	n/a	no-test	no-test	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
BEIJING-LCG2	T2D	FR	ACTIVE	100	3/3	0	no-test	98	3	99	1	online	online	online	online	online
BELLARMINE-ATLAS-T3	T3	n/a	ACTIVE	n/a	n/a	no-test	no-test	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
BNL-ATLAS	T1	US	ACTIVE	-	4/4	100	no-test	93	671	85	638	online	online	online	online	online
BNL-AWSEAST	T3D	n/a	ACTIVE	n/a	n/a	no-test	no-test	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
BNL-AWSWEST	T3D	n/a	ACTIVE	n/a	n/a	no-test	no-test	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
BOINC	T3D	n/a	ACTIVE	n/a	n/a	no-test	no-test	88	552	n/a	n/a	brokeroff	n/a	n/a	n/a	n/a
Brandeis-Atlas-T3	T3	n/a	ACTIVE	n/a	n/a	no-test	no-test	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
BU_ATLAS_Tier2	T2D	US	ACTIVE	-	4/4	100	no-test	99	7	91	36	online	online	online	online	online
CA-ALBERTA-WESTGRID-T2	T2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
CA-JADE	T3	CA	ACTIVE	-	n/a	no-test	no-test	no.data	0	n/a	n/a	online	n/a	online	online	online
CA-MCGILL-CLUMEQ-T2	T2D	CA	ACTIVE	100	4/4	100	0	99	35	87	14	online	online	online	online	online
CA-SCINET-T2	T2D	CA	ACTIVE	100	3/3	100	no-test	99	4	100	0	online	online	online	online	online
CA-VICTORIA-WESTGRID-T2	T2D	CA	ACTIVE	100	3/3	no-test	no-test	100	6	89	11	online	online	online	online	online
CERN-P1	T3	CERN	ACTIVE	-	n/a	no-test	no-test	no.data	0	n/a	n/a	online	n/a	online	online	online
CERN-PROD	T0	CERN	ACTIVE	100	n/a	no-test	no-test	99	11	100	0	online	brokeroff	online	online	blacklisted

Showing 1 to 160 of 160 entries DB query took 0.1402 s

(a)

(b)

Figure 4.10: An example of the *shifter view* in the SSB web interface [115].

The complexity of ATLAS require that the software must be highly robust and flexible in order to fulfil the needs of the experiment throughout its operational lifetime. There can indeed be changes in the physics goals and even in detector hardware. The software must also be able to cope with changes. The design and implementation of this kind of software is not an easy task. It should be possible to run on a distributed environment and it should fulfil the technical and the physics performance requirements.

Chapter 5

Monitoring

5.1 The Concept of Monitoring

One of the most promising perspective technologies of studying the social and natural processes is the system of monitoring. Nowadays, it is widely used in various spheres of human activity, such as economic, geographic, financial, information monitoring and others. Monitoring technology [120] is widely used, since the early 70s and immediately gained recognition. The monitoring technology became universal by allowing scientists to get the results of their research independently of the scale (global, regional, local, even personal). Depending on the areas of the research, the scientific literature indicates many types of monitoring.

Grid environments are typically distributed and consist of various complicated components. Monitoring of various components is required for different purposes. This can include status checking, performance tuning, troubleshooting and others. For example, a job has been submitted to the system. The particular job execution is supposed to be complete after approximately 15 min, but for some unknown reasons 3 hours have passed and the job is still not complete. The reasons for this can be different, for example problems with the job script, problems with the worker node where it is running, network problems, problems with a missing library, etc. Monitoring is required to determine the cause of issues within the system. Monitoring provides information for keeping track of the current state of work and helps track down problems. The following are main use cases of data monitoring:

- *Troubleshooting and detection of failures*: the reason of the failed job in the grid environment is difficult to determine. Detailed monitoring is needed for both online fault detection and for offline analysis.

- *Performance analysis and tuning*: usually grid applications face the problem of performance, such as unexpectedly low throughput or high latency. The determination of the performance problem requires detailed examination of related components.

- *Guiding scheduling decisions*: in the grid, the computational resources constantly change and the new measurements can be used for optimal or effective resource allocation and scheduling decisions. In this case, the resource monitoring is critical.

In the ATLAS experiment, monitoring systems are rather complicated. It is impossible to have only one monitoring tool for the huge collaborations using such a large-scale infrastructure. For this reason many monitoring systems have been designed and implemented. Each monitoring system uses its own architecture of presenting data values and is designed for monitoring a certain part of the experiment. The site administrators have to check different web sources and even operate grid commands manually in order to get the full information about the site status. However, the majority of monitoring systems can investigate the sites and their troubles in HEP experiments within the WLCG. This complicated procedure takes us to a new concept, called meta-monitoring.

5.2 Meta-Monitoring

Monitoring of the monitoring system is called meta-monitoring [121]. The meta-monitoring system detects situations where the monitoring system itself has a problem. The monitoring system needs to be more available and reliable than the services being monitored. Every monitoring system should have some kind of a meta-monitoring system. Even the smallest system needs a simple check to make sure that it is still running. By using a meta-monitoring system, it is possible to be aware of the completeness of a monitoring system. Meta-monitoring is the answer to the question: Is there something that is missing ? It is impossible to take into account every particular detail. Some aspects will always be overlooked. But with meta-monitoring it is possible to limit the unknown to a bare minimum. For larger systems the necessity of meta-monitoring system is more noticeable.

In ATLAS, for example, the PanDA Monitor and the DDM Dashboard have very complicated web interfaces as it includes all the information about the jobs and file transfers for all the WLCG sites. The extraction of the necessary information from the PanDA Monitor and the DDM dashboard takes time especially for a person who does not do it regularly. In order to have an overview about the status of the particular grid site, the navigation through numerous websites or a manual check through the Linux terminal is required. Of course, this causes some inconveniences, but on the other hand the multi-monitoring is often necessary to determine the main problems that are only visible by correlating smaller ones. This leads to the conclusion that the WLCG grid site status is spread across numerous WLCG websites and the usage of numerous monitoring tools leads to the need of having a meta-monitoring tool. The development of the meta-monitoring tool with a single web interface, which will display the most important information for the grid site is mandatory. For WLCG, an example of the meta-monitoring tool is HappyFace.

5.3 Monitoring and Meta-Monitoring Tools

Undoubtedly, the role of monitoring and meta-monitoring tools is very important to the correct functioning of the LHC. Monitoring and meta-monitoring is a necessary condition for sustainable development, as on the basis of research it is possible to get information about the current state and even predict future development projects.

The preferable components [122] that each meta-monitoring or monitoring system should have are the following:

- *Interface*: should provide an interface for accessing the information, for example a simple and generic API for querying information from other programs.

- *Single access point*: the final output should have a single interface, which shows all requested information from existing sources.
- *Fast accessibility*: provide fast access to the requested information from existing sources.
- *Modular structure*: the modular structure provides easy/visible output.
- *Up-to-date monitoring information*: the tool should provide fresh or up-to-date information.
- *History functionality*: should be able to show old data.
- *Comfortable usage*: the tool should be easy to use.
- *Extensible*: should be possible to extend the existing system and monitor new resources or data. Should be also possible to provide easy mechanisms for implementing such extensions.
- *Security*: should be possible to provide both secure connection to the monitoring system by using secure protocols and security in the level of user privileges.

In the grid, the resources can be changed frequently. The new resources and services can be added and removed. Moreover, grid resource and service properties can be changed as well. An example can be when a storage element is upgraded to a larger storage capacity, different access rates and protocols can be supported. These kind of dynamic changes can make both the resource discovery in terms of finding the appropriate resources to fulfil the task, and the resource monitoring, in terms of observing resources or services to keep track of the status change for fixing problems.

The Globus Toolkit solves this problem with the MDS [123] which is a collection of components implemented to monitor and discover resources and services.

The HappyFace project is my research topic and will be discussed in detail in the next chapter.

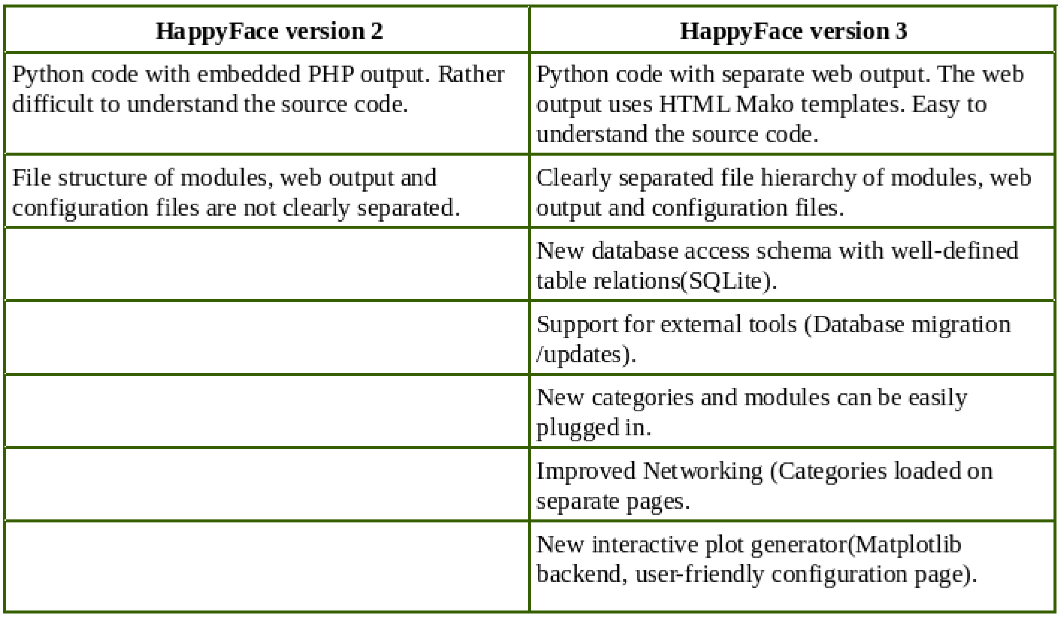
Chapter 6

The HappyFace Project

6.1 Description of HappyFace

6.1.1 Introduction

The HappyFace project [124] is a joint collaboration between Georg-August-University Göttingen and Karlsruhe Institute of Technology (KIT). The first version of HappyFace (HappyFace 1.0) was designed by KIT in 2008. In 2009, HappyFace 2.0 was released, which was relying on a more sophisticated and generic approach. The current version of HappyFace is HappyFace version 3, which was released in 2013 and is fully replacing HappyFace version 2. HappyFace version 2 and version 3 are very different from each other in terms of source code structure and development. The main differences between HappyFace version 2 and version 3 is shown in Figure 6.1.



HappyFace version 2	HappyFace version 3
Python code with embedded PHP output. Rather difficult to understand the source code.	Python code with separate web output. The web output uses HTML Mako templates. Easy to understand the source code.
File structure of modules, web output and configuration files are not clearly separated.	Clearly separated file hierarchy of modules, web output and configuration files.
	New database access schema with well-defined table relations(SQLite).
	Support for external tools (Database migration /updates).
	New categories and modules can be easily plugged in.
	Improved Networking (Categories loaded on separate pages.
	New interactive plot generator(Matplotlib backend, user-friendly configuration page).

Figure 6.1: The main differences between HappyFace version 2 and version 3.

The development of new modules takes places at Georg-August-University Göttingen, KIT and Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen University. Besides KIT and the University of Göttingen other universities such as RWTH Aachen University, the research center of DESY and the High Energy Accelerator Research Organisation (KEK) are using HappyFace for their purposes.

Initially, HappyFace was designed as a meta-monitoring tool that aggregates, processes and stores information from various monitoring tools into a single web interface. Nowadays, HappyFace is not only a meta-monitoring tool, but also a monitoring tool that aggregates information from the grid system directly.

HappyFace has a modular structure where each module is responsible for extracting, processing and storing data from defined WLCG monitoring sources.

The modular structure gives several advantages:

- It supports *agile software development style* [125]. This is very important as many developers can work on different modules simultaneously.
- Edited modules can be easily deployed and in case of failures the previous version can be restored simply.
- New modules can easily be plugged in and in case of failures will not harm the work of the whole project, but only the corresponding problematic module.
- It makes the project more flexible, as WLCG sources and their corresponding modules might be changed.

The HappyFace web interface displays all information that is stored in the HappyFace database using HTML Mako template [126]. In terms of the database, HappyFace uses a SQLite database [127]. The usage of that database enables history functionality for the HappyFace system and stored information can be accessed at any time. The core system is called HappyFaceCore. It includes individual modules, web output and configuration files. HappyFaceCore also contains certain functions that are necessary to download the content from a certain web interface, to run the software and to enable the Python *plotgenerator* [128]. HappyFace is flexible, easy configurable and can be adapted to site-specific requirements. In the following sections, the HappyFace workflow, the web interface, the category/module configuration and the database is described in detail.

6.1.2 Basic Workflow

The workflow of HappyFace is shown in Figure 6.2.

For each category, there can be an unlimited number of modules. All necessary details about these categories/modules are mentioned in specific configuration files. There exists exactly one configuration file per category/module.

For each module there are three main files, which are a Python script, an HTML file and a config file that needs to be created before in order to have a running module.

The *acquire.py* Python script downloads data, parses data and stores them in the database. The *acquire.py* Python script is periodically executing every 15 minutes, e.g. via a cron job [129] under Linux/Unix operating systems.

The *render.py* Python script retrieves data from the database and renders it to the HTML template file.

Each module has an implemented function that accesses the previously stored data in the HappyFace database. The render script executes this function of each module and provides the extracted data to the module HTML template file.

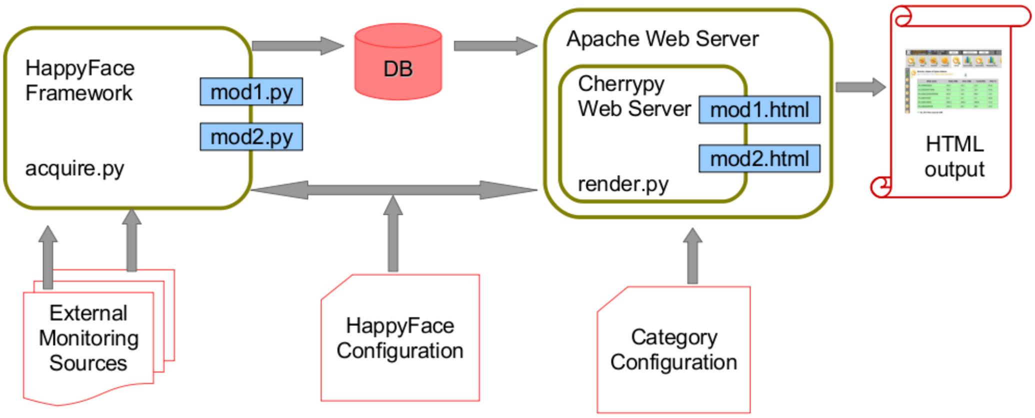


Figure 6.2: The workflow of HappyFace. Taken from [122].

Each module can have only one table and unlimited number of subtables. However, if there are changes in the database structure, the whole database schema needs to be updated. This is done by using the *tools.py* Python script.

The full cycle of HappyFace is the following:

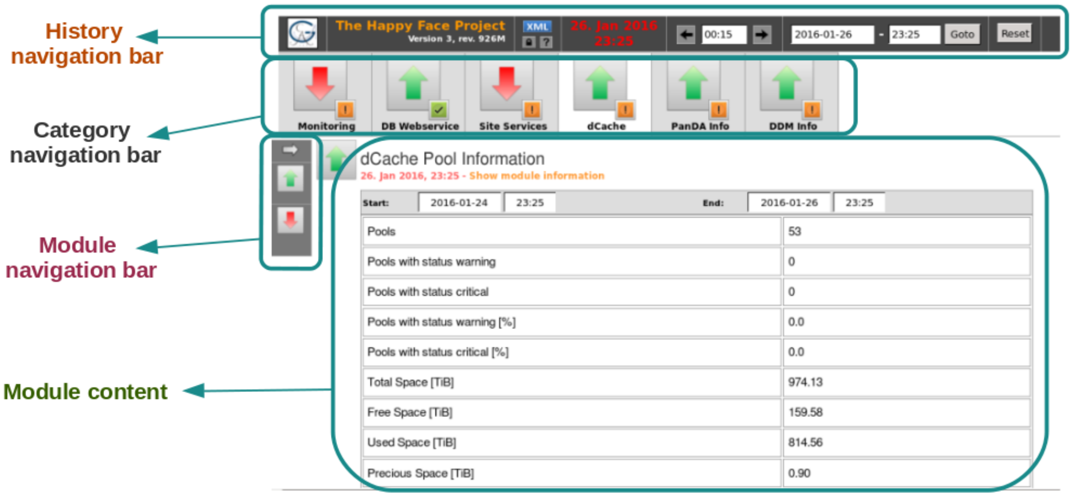
- initializing modules with configuration parameters,
- downloading necessary sources (mentioned in the configuration files),
- aggregating, extracting necessary data from downloaded sources,
- storing time-stamped data in HappyFace database (HappyFace.db),
- displaying stored data on the HappyFace web interface.

6.1.3 Web Interface

The HappyFace web interface is divided into several navigation bars (Figure 6.3), which are history, category, module navigation bars and module content.

By changing the date/time in the history navigation bar, the stored information from the HappyFace database will be retried and displayed on the module content. 15 minutes is the default value of the time interval in the history of navigation as well as for running the *acquire.py* Python script.

Access to the categories and its modules can be *open*, *permod*, or *restricted*. It depends on the parameters specified in the category configuration file. As a default value, it is specified *open*, which allows full access for the user to any category/module. If *permod* is specified in the category configuration file, then the category will be open but some of the modules would be restricted. If *restricted* is specified - only authorised users can have an access to the entire category/module. The category navigation bar has a simple rating system (see Section 6.1.4) through which it reacts to any module status change and displays the corresponding icon on the web interface. This kind of modular structure is very intuitive, as with one look it is possible to get an overview about the ongoing issues on the site.



dCache Pool Information

26. Jan 2016, 23:25 - Show module information

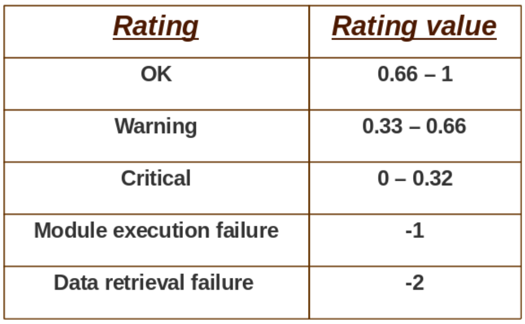
Start: 2016-01-24 23:25 End: 2016-01-26 23:25

Pools	53
Pools with status warning	0
Pools with status critical	0
Pools with status warning [%]	0.0
Pools with status critical [%]	0.0
Total Space [TiB]	974.13
Free Space [TiB]	159.58
Used Space [TiB]	814.56
Precious Space [TiB]	0.90

Figure 6.3: HappyFace web interface.

6.1.4 Rating System

HappyFace has a very simple rating system. Each module has a rating value (Figure 6.4).



Rating	Rating value
OK	0.66 – 1
Warning	0.33 – 0.66
Critical	0 – 0.32
Module execution failure	-1
Data retrieval failure	-2

Figure 6.4: HappyFace rating system.

Depending on the rating value, the corresponding icon (coloured arrow) is displayed on the web interface. If the rating value is between 0 and 0.33, the status is *critical* and the arrow will turn red and will point down. If the rating value is between 0.33 and 0.66, then the status is *warning* and the arrow will be yellow and horizontal. If the rating value is between 0.66 and 1, then the status is *ok* and the arrow will be green and will point up. If, for some reason, the module execution fails, then the value is set to -1 and if the data retrieval from the database fails then it is set to -2. In case of both negative values at the same time, the icon will contain a small exclamation mark. Usually in every category there are several modules. In this case, the overall category rating value depends on the rating value of the existing modules in that particular category.

The category rating value is calculated by one of the following algorithms: *unrated*, *worst* and *average*. A rating algorithm is defined in the category configuration file. If the algorithm *unrated* is set in the configuration file, then modules status will not be considered. If the algorithm *worst* is set in the configuration file, then the minimum value over all modules status will be taken. And last, if the algorithm *average* is set in the configuration file, then the average value over all modules status will be taken.

6.1.5 Database Structure

The HappyFace database has three main tables, *module_instances*, *hf_runs* and *mod_className* (Figure 6.5).

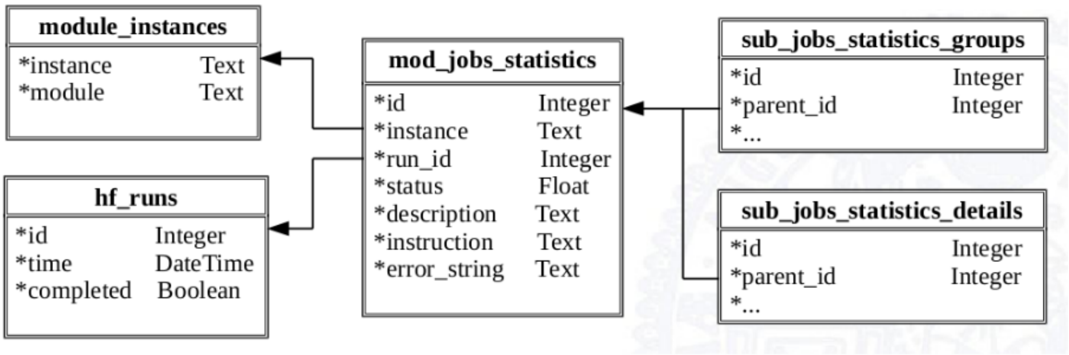
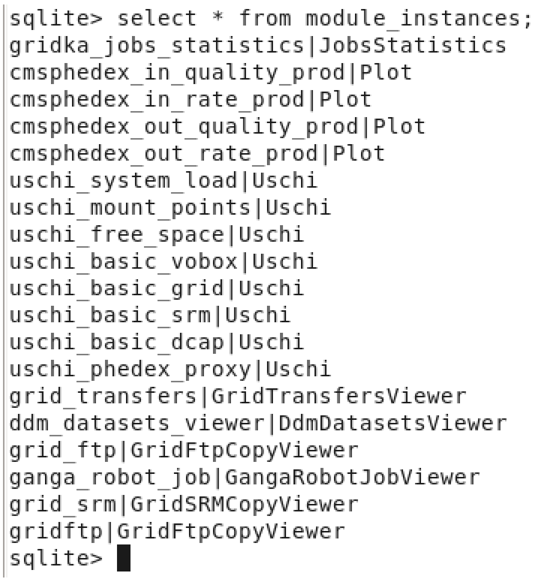


Figure 6.5: HappyFace database schema.

The table *module_instances* stores information about the name of the module and connected instances with it as shown in Figure 6.5. Each module can have several instances (Figure 6.6).



```
sqlite> select * from module_instances;
gridka_jobs_statistics|JobsStatistics
cmsphedex_in_quality_prod|Plot
cmsphedex_in_rate_prod|Plot
cmsphedex_out_quality_prod|Plot
cmsphedex_out_rate_prod|Plot
uschi_system_load|Uschi
uschi_mount_points|Uschi
uschi_free_space|Uschi
uschi_basic_vobox|Uschi
uschi_basic_grid|Uschi
uschi_basic_srm|Uschi
uschi_basic_dcap|Uschi
uschi_phedex_proxy|Uschi
grid_transfers|GridTransfersViewer
ddm_datasets_viewer|DdmDatasetsViewer
grid_ftp|GridFtpCopyViewer
ganga_robot_job|GangaRobotJobViewer
grid_srm|GridSRMCopyViewer
gridftp|GridFtpCopyViewer
sqlite> 
```

Figure 6.6: HappyFace *module_instances* table content.

The table *hf_runs* stores information about the status of the executed *acquire.py* Python script, i.e. id, time and a boolean flag that indicates success or fail execution.

The table *mod_className* stores the general information about the module. The name of the table makes up automatically by adding to *mod_* the module *className* with "_" i.e. if the module class name is *JobsStatistics* then the table name will be *mod_job_statistics*. The table *mod_className* stores general information about the module such as a *status*, *error message*, *description*, etc.

Each module can have multiple subtables. The columns of the subtables are defined manually in the Python script of each module. Each time when *acquire.py* Python script is executed, one or many rows are inserted into the existing module's tables and subtables and one row is inserted into the *hf_runs* table.

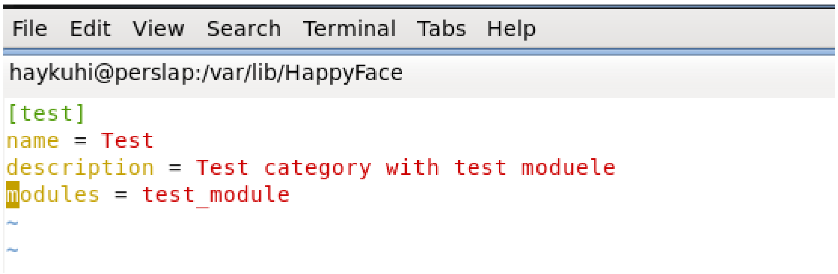
6.1.6 Module Development

The module development in HappyFace is straightforward unless the structure of the source code is unknown.

Each HappyFace module consists of four main files: two configuration files (one for the category and one for the module), a Python script and a HTML template. In order to plug the module into the framework mentioned above, the structure needs to be followed up.

The module *Test* is a working module and is presented here as an example in order to understand the workflow of HappyFace on the level of source coding. It is successfully pluged into the HappyFace framework and is running without any errors.

The category configuration file (*test.cfg*) content for the Test category is shown in Figure 6.7. The category configuration file is located at the */var/lib/HappyFace/config/categories-enabled* path.


File Edit View Search Terminal Tabs Help
haykuhi@perslap:/var/lib/HappyFace

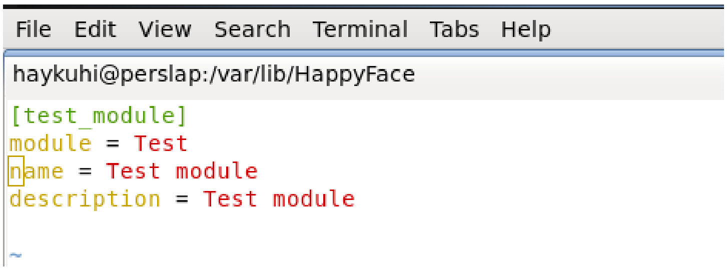
```
[test]
name = Test
description = Test category with test moduele
modules = test_module
~
~
```

Figure 6.7: The category configuration file for the *Test* category.

- *name* – contains the name of the category that will appear in the web interface.
- *description* – contains the description of the category for what it will be used.
- *modules* – contains the associated module(s) with the category.

The module configuration file (*test.cfg*) content for the *Test* module is shown in Figure 6.8. The module configuration file is located at the */var/lib/HappyFace/config/modules-enabled* path.

- *module* – contains the name of the Python script that is associated with that particular module.
- *name* – contains the name of the module that will appear in the web interface.
- *description* – contains the description of the module(s) for what it/they will be used.



haykuhi@perslap:/var/lib/HappyFace

```
[test_module]
module = Test
name = Test module
description = Test module

~
```

Figure 6.8: The module configuration file for the *Test* module.

```python
import hf, lxml, logging, datetime
from sqlalchemy import *

class Test(hf.module.ModuleBase):
    config_keys = {
        'site_keys': (' ', ' ')
    }
    config_hint = '  '

    table_columns = [], []

    subtable_columns = {'details': ([
            Column("test", TEXT),
            Column("status", TEXT)
    ], []),}

    def prepareAcquisition(self):

        # definition of the database table keys and pre-defined values
        self.details_db_value_list = []


    def extractData(self):
        data = {}
        data['status']=1

        details_db_value = {}
        details_db_value['test'] = "Test"
        details_db_value['status'] = "OK"
        self.details_db_value_list.append(details_db_value)

        return data

    def fillSubtables(self, parent_id):
        self.subtables['details'].insert().execute([dict(parent_id=parent_id, **row)
        for row in self.details_db_value_list])

    def getTemplateData(self):
        data = hf.module.ModuleBase.getTemplateData(self)

        list = self.subtables['details'].select().where(
        self.subtables['details'].c.parent_id==self.dataset['id']).execute().fetchall()

        data['details_list'] = map(dict, list)

        return data
```

Figure 6.9: The module Python script for the *Test* module.

The module Python script (*Test.py*) content for the *Test* module is shown in Figure 6.9. The module Python script is located at the */var/lib/HappyFace/modules* path.

All module classes are inherited from the main *hf.module.ModuleBase* class, which determines the basic structure of all modules. The *config_keys* are variables that will be read by the *ConfigParser* in the script.

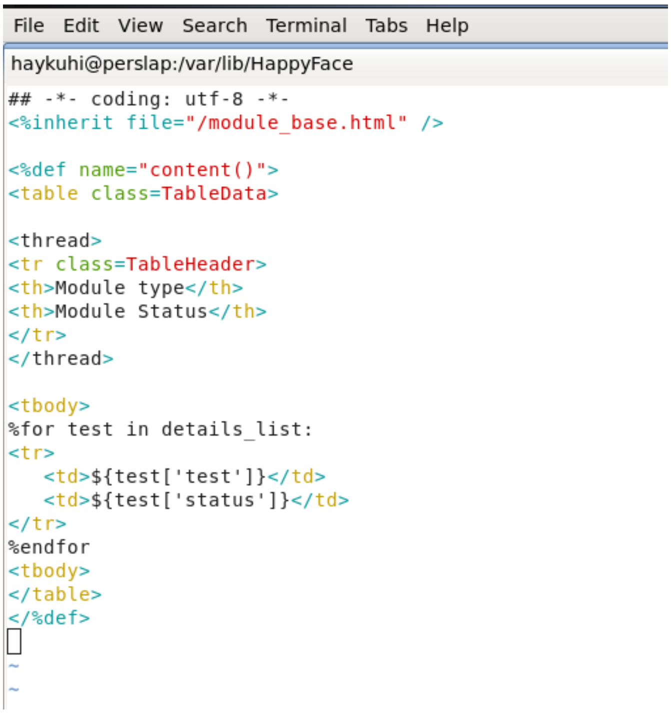
Each module has one default table (*table_columns*). If there are no other columns set in *table_columns*, then it will have the default table structure (Figure 6.5).

As it was mentioned before, each table can have unlimited number of subtables. Subtables meant for storing additional information, which is not stored in the default module table. In this example, the *subtable_columns* table has two columns: *test* and *status*.

The Python script contains four main defined functions. To create new modules the implementation of these functions is mandatory.

- *prepareAcquisition()* - used for commissioning, monitoring data downloads,
- *extractData()* - used for the extraction of necessary information from the previously downloaded files,
- *fillSubtables()* - used for inserting the extracted data in the HappyFace module subtable(s),
- *getTemplateData()* - used for creating variables , which can be accessed from the HTML mako template.

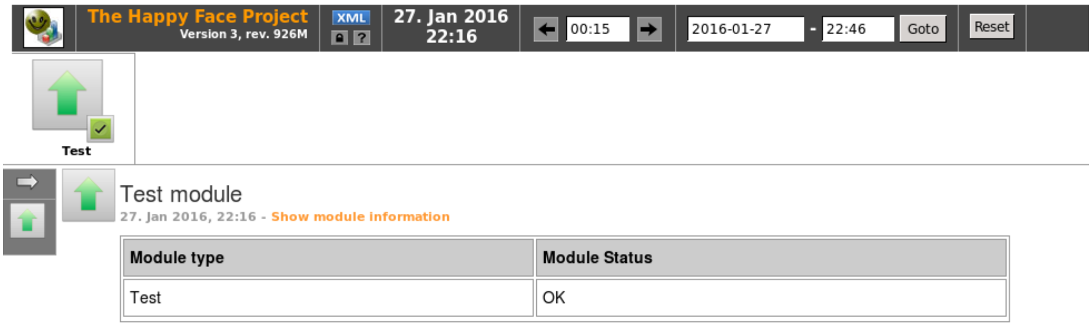
The module HTML template file (*Test.html*) content for the *Test* module is shown in Figure 6.10. The module HTML template file is located at the */var/lib/HappyFace/modules* path.


File Edit View Search Terminal Tabs Help
haykuhi@perslap:/var/lib/HappyFace

```
## -*- coding: utf-8 -*-
<%inherit file="/module_base.html" />

<%def name="content()">
<table class=TableData>

<thread>
<tr class=TableHeader>
<th>Module type</th>
<th>Module Status</th>
</tr>
</thread>

<tbody>
%for test in details_list:
<tr>
   <td>${test['test']}</td>
   <td>${test['status']}</td>
</tr>
%endfor
<tbody>
</table>
</%def>
```

Figure 6.10: The module HTML template for the *Test* module.

By running the *acquire.py* Python script, the module is executed and the output is displayed on the web interface. For the *Test* module web interface is shown in Figure 6.11.



Test module

27. Jan 2016, 22:16 - Show module information

Module type	Module Status
Test	OK

Figure 6.11: The web interface for the *Test* module.

As we can see the output of the *Test* module is a simple table that displays the information stored in the HappyFace database.

6.2 HappyFace for the ATLAS Experiment

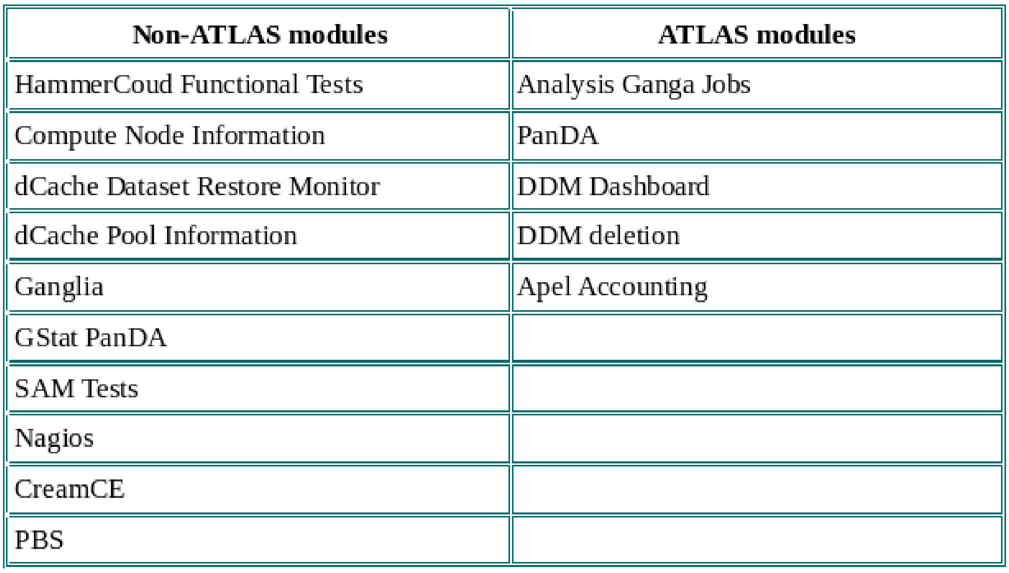
HappyFace does a clear separation between HappyFace core modules and individual modules. Each module retrieves its specific information from external source(s), to process and display the relevant information on the web interface.

The Tier-2 GoeGrid center, which is located at GWDG in Göttingen (see Section 3.5) is one of the ATLAS Tier-2 centers. Initially, the HappyFace project was designed and developed mainly in Karlsruhe Institute of Technology (KIT). KIT is involved in the LHC CMS experiment, which means that HappyFace modules were designed and developed mainly for the CMS experiment. Any other sites, for example the GoeGrid, which would like to use HappyFace, needs to install and adapt a HappyFace instance according to its own site requirements. Due to the fact that once some of the modules are useless for the ATLAS experiment, they need to be removed or to be fully rewritten. In some particular cases there is a need to redevelop the whole module from the beginning.

So far a number of different non-ATLAS and ATLAS modules have been designed and developed by the Göttingen group members. The list of developed modules is given in the Figure 6.12.

All developed modules are generic and are easily adaptable to any ATLAS grid site in the WLCG.

More detailed information about above mentioned modules can be found in the following works [130], [131], [132].



Non-ATLAS modules	ATLAS modules
HammerCoud Functional Tests	Analysis Ganga Jobs
Compute Node Information	PanDA
dCache Dataset Restore Monitor	DDM Dashboard
dCache Pool Information	DDM deletion
Ganglia	Apel Accounting
GStat PanDA	
SAM Tests	
Nagios	
CreamCE	
PBS	

Figure 6.12: The list of developed modules by the Göttingen group members.

6.3 Grid-Enabled HappyFace

6.3.1 The Concept and Design

By summarising section 6.1, HappyFace aggregates, processes and stores the information and the status of various HEP monitoring resources into the common database of HappyFace. It displays the information and the status through a single interface. However, this model of HappyFace relied on the monitoring resources which are always under development in the HEP experiments (Figure 6.13). This made the system unstable.

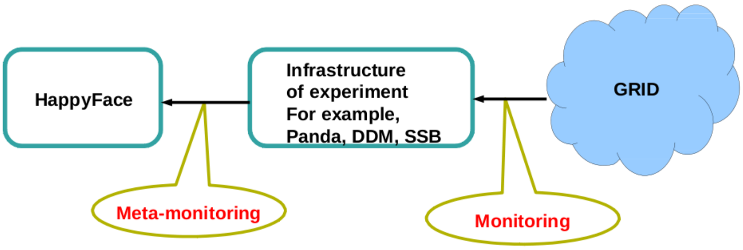


Figure 6.13: HappyFace initial model. Taken from [122].

Accordingly, HappyFace needs to have direct access methods to the grid applications and grid service layers. To cope with this issue, we use a reliable HEP software repository, the CernVM File System (CVMFS) [133], [134], supplying grid environments and available HEP tools.

We propose a new implementation and an architecture of HappyFace, the so-called grid-enabled HappyFace. It allows its basic framework to connect directly to the grid user applications and the grid collective services, without involving the monitoring resources in the HEP grid systems (Figure 6.14).

This approach gives HappyFace several advantages:

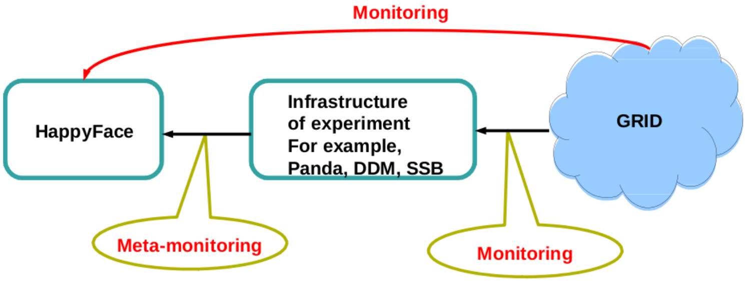


Figure 6.14: HappyFace modified model. Taken from [122].

- *Portability*: providing an independent and generic monitoring system among the HEP grid systems.
- *Functionality*: allowing users to perform various diagnostic tools in the individual HEP grid systems and grid sites.
- *Flexibility*: making HappyFace beneficial and open for the various distributed grid computing environments.

These factors need to be taken into consideration when implementing new extensions in HappyFace to accept new monitoring resources described above.

The Göttingen group members designed the grid-enabled HappyFace as an extension of the HappyFace system. In order to have access to the grid system there are certain requirements:

- Proper X.509 certificate
- Integration with grid environments and available HEP tools
- New source code implementation as an extension

The grid-enabled extension is based on the concept of object-oriented programming (OOP) (see Section 6.3.1.1) and uses design patterns (see Section 6.3.1.2) for structuring the classes and objects.

An implementation of the new extension is described in detail in Section 6.3.2.

6.3.1.1 Object-Oriented Programming (OOP)

Object-oriented programming (OOP) [135] refers to a software programming that is based on the concept of *objects*. It gives a possibility to create manually the data structure that can contain a collection of different types of variables (*attributes*) and functions (*methods*). This kind of data structure is called *a class*. The objects are instances of classes that typically determine their types.

OOP is supported by almost all main programming languages (Python, C++, Delphi, Java, C, Perl, Ruby, PHP, etc.).

OOP is based on the three main concepts:

- *Encapsulation:* it binds together data attributes and methods. In addition, the internal representation of an object is generally hidden from the external object definitions. Typically, only an object's own methods can directly inspect or manipulate data attributes.

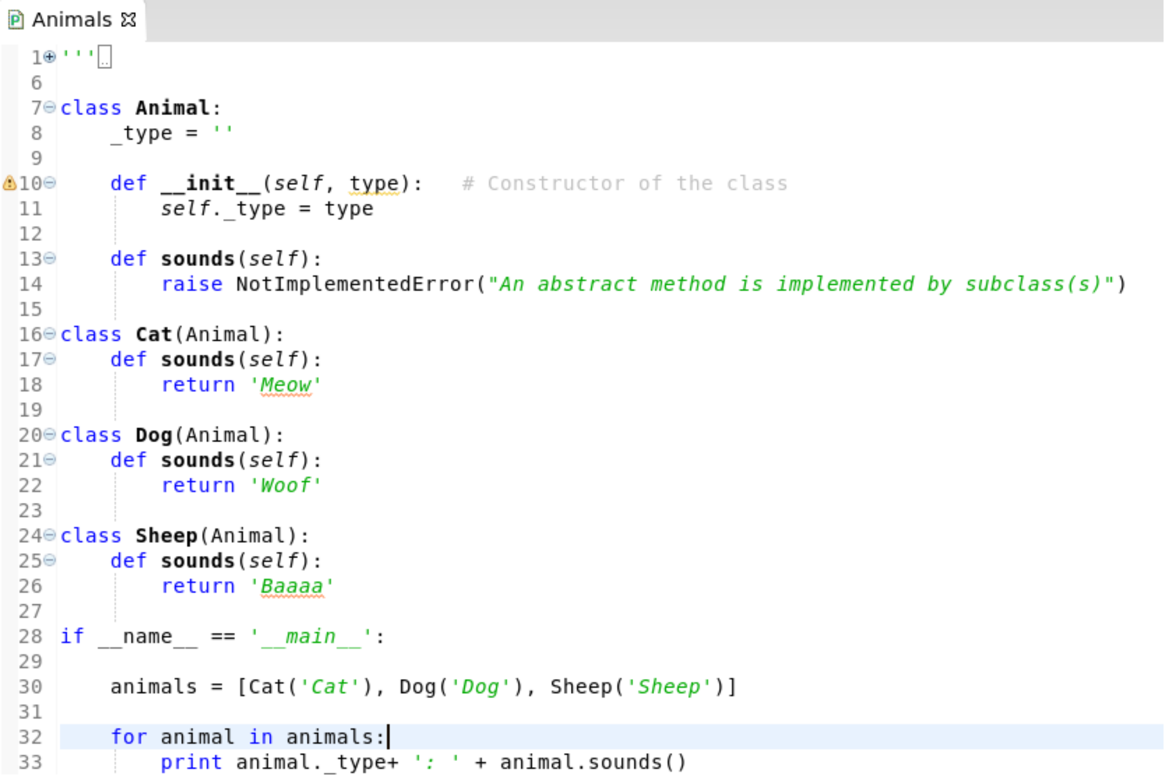
  For example, in Java language the class attributes and methods can have access modifiers such as *private*, *public* and *protected*. The access modifiers restrict the access to the defined attributes or methods.

  - *private*: only the current class has access to the attributes and methods.
  - *protected*: only the current class and sub-classes have access to the attributes and methods.
  - *public*: any class can refer to the attributes and call the methods.

  In Python, the private methods have names that start with an underscore.

- *Inheritance:* a feature that represents the relationship between different classes. This allows classes to be arranged in a hierarchical way. For example, *class A* can inherit from *class B*. This means that all the attributes and methods existing in *class B* will appear in *class A*. In addition, *class A* can also have its own attributes and methods. Sub-classes are also allowed to override the methods defined in the superclass (parent class).

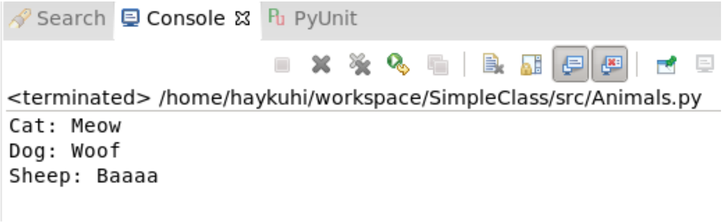
- *Polymorphism:* is an ability to handle object differently according to the data type or class. An example is shown in Figure 6.15.



```
'''

class Animal:
    _type = ''

    def __init__(self, type):    # Constructor of the class
        self._type = type

    def sounds(self):
        raise NotImplementedError("An abstract method is implemented by subclass(s)")

class Cat(Animal):
    def sounds(self):
        return 'Meow'

class Dog(Animal):
    def sounds(self):
        return 'Woof'

class Sheep(Animal):
    def sounds(self):
        return 'Baaaa'

if __name__ == '__main__':

    animals = [Cat('Cat'), Dog('Dog'), Sheep('Sheep')]

    for animal in animals:
        print animal._type+ ': ' + animal.sounds()
```

Figure 6.15: An example of the source code that shows the class inheritance and polymorphism.

Depending on the class object method, the corresponding *voices()* method will be called. The output is shown in Figure 6.16.



```
<terminated> /home/haykuhi/workspace/SimpleClass/src/Animals.py
Cat: Meow
Dog: Woof
Sheep: Baaaa
```

Figure 6.16: The output of the Python script.

The main benefit of polymorphism is that it simplifies the programming interface. It supports the overriding of methods, which stands for the use of the method's name multiple times. This means that instead of inventing a new name for each new method, the same name can be easily reused.

6.3.1.2 Design Patterns

Designing an optimal or an effective OOP from first try is a very difficult and almost impossible task. To define class interfaces, relationships between them and necessary methods in a way that they would not be modified or redesigned later is impossible. It is even more difficult to make them general and reusable. The design of the class should be generic enough to solve future problems or fulfil future requirements, but at the same time specific enough, to solve a particular problem.

Programmers typically design classes some way, then reuse them, thus modifying them multiple times. Before the design of classes can be considered final, it must be tested. Modifying the code can be a tedious task.

Design patterns [136] in computer science are a formal way to document solutions to common problems at the beginning of the design of any application or system. The design patterns offer general reusable solutions that can be used multiple times in different situations. They help to minimise or isolate the endless iterations or modifications over class attributes and methods. The design patterns offer a concept to make a clear separation between the same and various class characteristics. They offer to generalise parts of the classes, which can be reused anytime and some of them are kept specific so that they can be used occasionally. This way of separation is very flexible and minimises the changes in the whole source code. The goal of design patterns is to reduce changes in the source code.

Design patterns are usually documented by using Unified Modeling Language (UML) tools [137]. UML is a common modeling language used in the field of software development for describing or visualising the structure or design of a system.

In this thesis, the UML diagrams are designed using the *Umbrello UML Modeller* tool [138]. It is an open-source tool available for Microsoft Windows and Unix-like platforms.

A class diagram is a static type of UML structure diagram that describes class attributes, methods and relationships between them.

In the UML diagram, classes are displayed in boxes that includes three sections: class names, class attributes, class methods, see Figure 6.17.

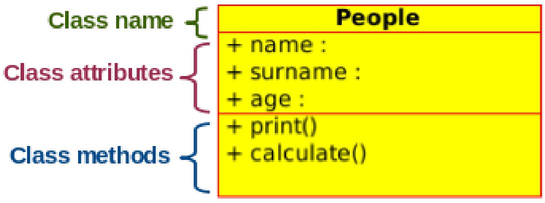


Figure 6.17: A class sections.

The relationships between classes describe the logical connections between them. UML keep up the following relationships, see 6.18.

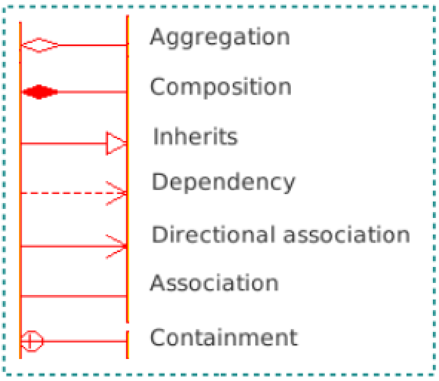


Figure 6.18: UML relations notation

The class inheritance can be seen as a design pattern. For example, one needs to design a *Bear* simulation game. An example is shown in the *Bear* class. Initially, the *Bear* class will be designed as it is shown in Figure 6.19. There will be one superclass named *Bear* and two sub-classes called *PolarBear* and *BrownBear*.

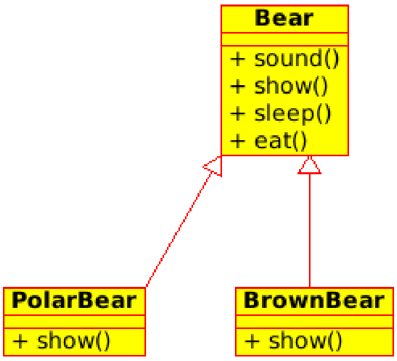


Figure 6.19: The UML diagram of the *Bear* class with two sub-classes (*PolarBear and BrownBear*).

The initial model is correct unless there will be an update and another type of *bear* will appear, for example *TeddyBear* (Figure 6.20).

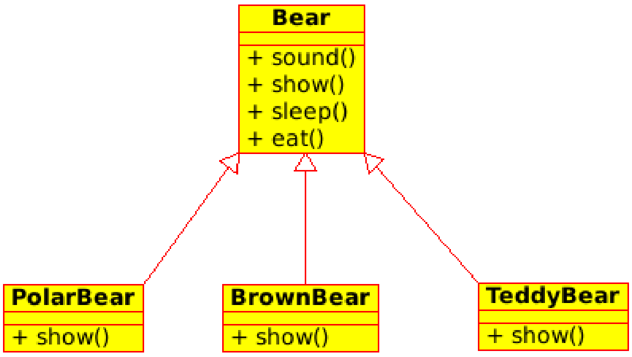


Figure 6.20: The UML diagram of the *Bear* class with three sub-classes (*PolarBear, BrownBear, TeddyBear*).

According to the designed model, the *TeddyBear* sub-class is able to eat and to sleep, which is not correct. The newly appeared sub-class *(TeddyBear)* does not fit the designed class model and pushes modifications in the initial model itself.

After modifications, the UML diagram will look like the following (Figure 6.21). There should be one base class (*Bear*) from which all other sub-classes (*PolarBear, BrownBear, TeddyBear*) should inherit and two interfaces (*Sleeping*, *Eating*). These two interfaces must be implemented by the sub-classes on demand.

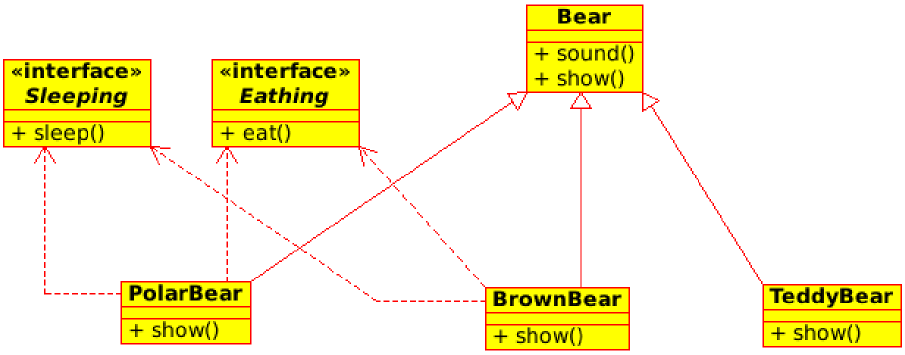


Figure 6.21: The final UML diagram of the *Bear* class with its sub-classes (*PolarBear, BrownBear, TeddyBear*).

To modify the initial model by taking into account that not all bears are able to eat and to sleep is logically correct.

There are more than 20 different design patterns which are classified over three types.

- *Creational*: describe the best way of creating an object. An example of such a pattern is a *singleton class* where only a single instance of a class can be created. This type of pattern is used when no more than one class instance can be created.
- *Structural*: describes the relationship between objects and classes that are satisfying a particular system requirement in a way that future changes in the system will not require changes in already existing class relations. The *Bear* class, which was described above, is an example of the structural type of design pattern.

- *Behavioral*: describe the interactions between objects. The idea is to encapsulate various parts in a way that the changes in one source code will not affect the rest.

As is the case with all concepts, design patterns have advantages and disadvantages.

Advantages of using design patterns:

- offer standard solutions to common programming problems
- highly flexible and can be used for practically any type of applications
- design and implementation of the structure reaches a certain goal
- increase the quality of the code
- make connections between program components
- language independent and can be applied to any language that supports OOP

Disadvantages of using design patterns:

- does not lead to direct code reuse
- complex in its nature
- need programming experience and discussions
- decrease understandability by increasing the amount of source codes
- consumes more memory because of their generalised formats

Despite their complexity, design patterns are very worth to use. They allow making high level source codes, which is an extremely important factor in software programming.

Grid-enabled HappyFace is designed based on the concept of OOP with using design patterns.

6.3.2 Implementation

With the new grid-enabled extension, the workflow of HappyFace looks as it is displayed in Figure 6.22.

The idea of the new workflow is providing access to the grid from the HappyFace system itself. This is done by setting up the necessary environment, by using the proper X.509 certificate, the HappyFace configuration file and the grid subprocess class (an extension of the Python subprocess class) from the HappyFace system.

Access methods are used for

- retrieving needed information from the grid,

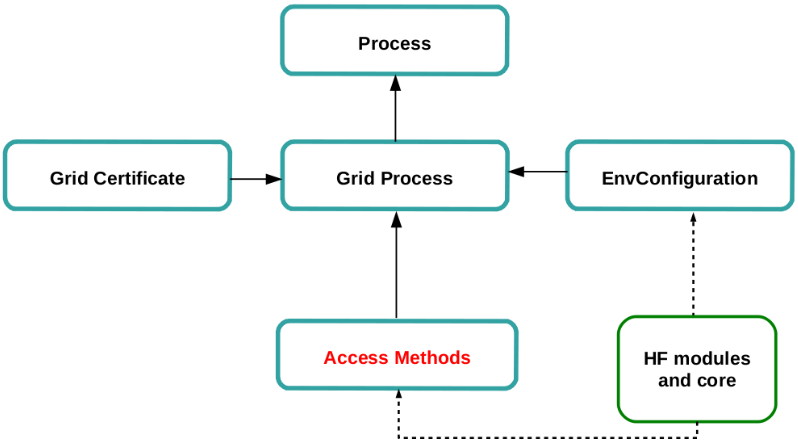


Figure 6.22: The grid-enabled HappyFace workflow. Taken from [122].

- storing them into the HappyFace database,
- displaying them on the HappyFace web interface.

Access methods are similar to HappyFace modules/categories, but with the grid part included in it.

The grid-enabled HappyFace extension consists of two main packages: *GridEngine* and *GridToolkit*.

GridEngine consists of three Python scripts: *envreader.py*, *gridcertificate.py*, *gridsubprocess.py*. All these classes are connected with each other.

The structure of classes is the following:

- The *envreader.py* Python script consists of BaseEnv, GridEnv, CvmfsEnv, BaseEnvReader, GridEnvReader, CvmfsEnvReader classes and four functions, which are described below. The *envreader.py* script file is used for setting up the grid and the cvmfs environments by reading necessary variables from the HappyFace configuration file.

  The UML diagram of the *envreader.py* Python script classes is displayed in Figure 6.23.

  - The *getuid()* function is used for getting user id information, which will be used later by the GridEnv class for setting up the grid environment.
  - The *load_happyface_config_reader()* function is checking if the HappyFace configuration file is available. This function is later used by the BaseEnvReader class.
  - The *check_if_happyface_config_key_exist()* function is used for checking if the HappyFace configuration file structure is followed. The function is later used by the BaseEnvReader class. There is a defined HappyFace configuration file structure, which is displayed in Figure 6.24.

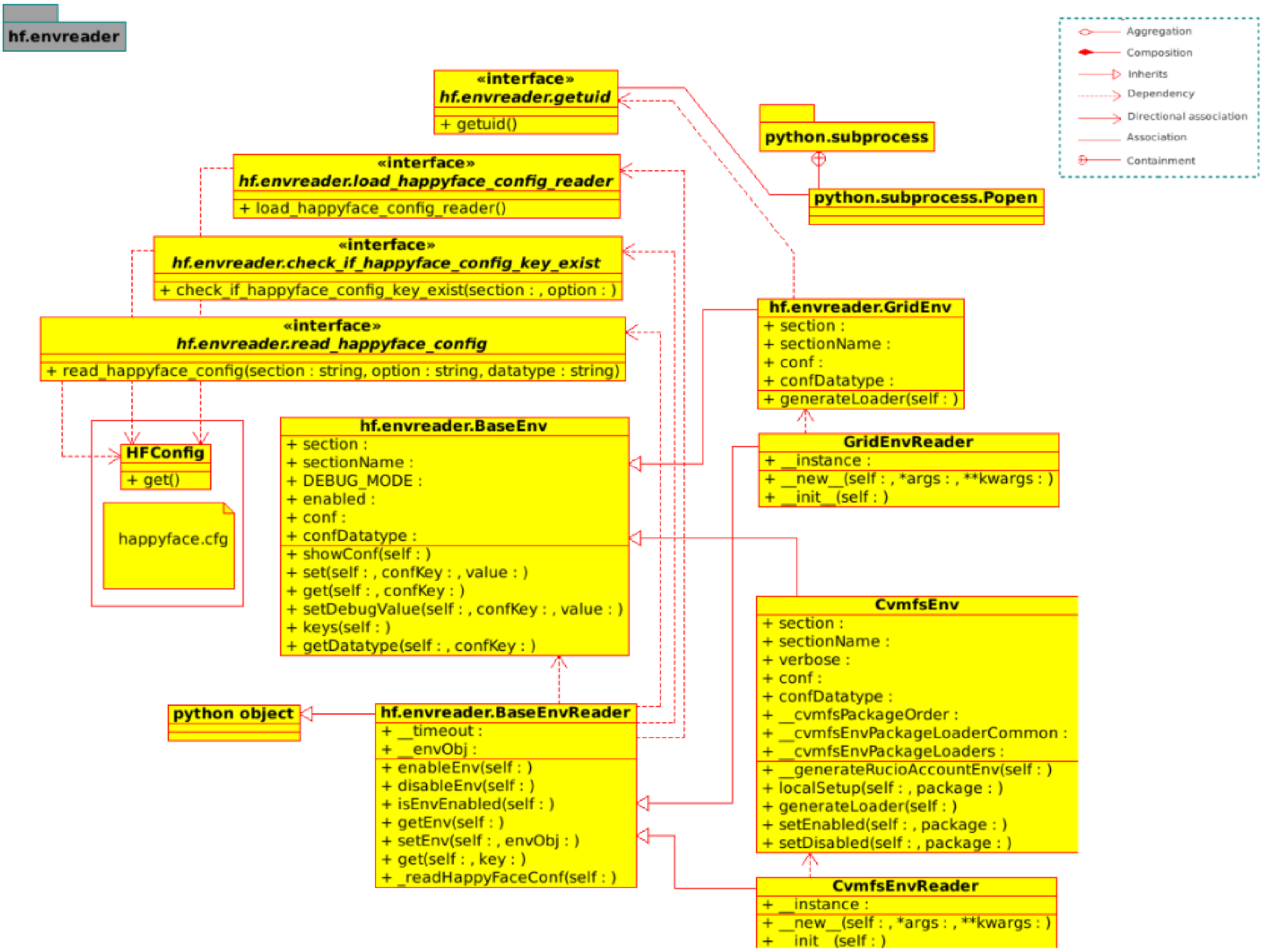

hf.envreader
Aggregation
Composition
Inherits
Dependency
Directional association
Association
Containment
«interface»
hf.envreader.getuid
+ getuid()
python.subprocess
python.subprocess.Popen
«interface»
hf.envreader.load_happyface_config_reader
+ load_happyface_config_reader()
«interface»
hf.envreader.check_if_happyface_config_key_exist
+ check_if_happyface_config_key_exist(section : , option :)
«interface»
hf.envreader.read_happyface_config
+ read_happyface_config(section : string, option : string, datatype : string)
hf.envreader.GridEnv
+ section :
+ sectionName :
+ conf :
+ confDatatype :
+ generateLoader(self :)
HFConfig
+ get()
happyface.cfg
hf.envreader.BaseEnv
+ section :
+ sectionName :
+ DEBUG_MODE :
+ enabled :
+ conf :
+ confDatatype :
+ showConf(self :)
+ set(self : , confKey : , value :)
+ get(self : , confKey :)
+ setDebugValue(self : , confKey : , value :)
+ keys(self :)
+ getDatatype(self : , confKey :)
GridEnvReader
+ __instance :
+ __new__(self : , *args : , **kwargs :)
+ __init__(self :)
CvmfsEnv
+ section :
+ sectionName :
+ verbose :
+ conf :
+ confDatatype :
+ __cvmfsPackageOrder :
+ __cvmfsEnvPackageLoaderCommon :
+ __cvmfsEnvPackageLoaders :
+ __generateRucioAccountEnv(self :)
+ localSetup(self : , package :)
+ generateLoader(self :)
+ setEnabled(self : , package :)
+ setDisabled(self : , package :)
python object
hf.envreader.BaseEnvReader
+ __timeout :
+ __envObj :
+ enableEnv(self :)
+ disableEnv(self :)
+ isEnvEnabled(self :)
+ getEnv(self :)
+ setEnv(self : , envObj :)
+ get(self : , key :)
+ _readHappyFaceConf(self :)
CvmfsEnvReader
+ __instance :
+ __new__(self : , *args : , **kwargs :)
+ __init__(self :)

Figure 6.23: The UML diagram of the *envreader.py* Python script classes.

[section]

variable1 = value1

variable2 = value2

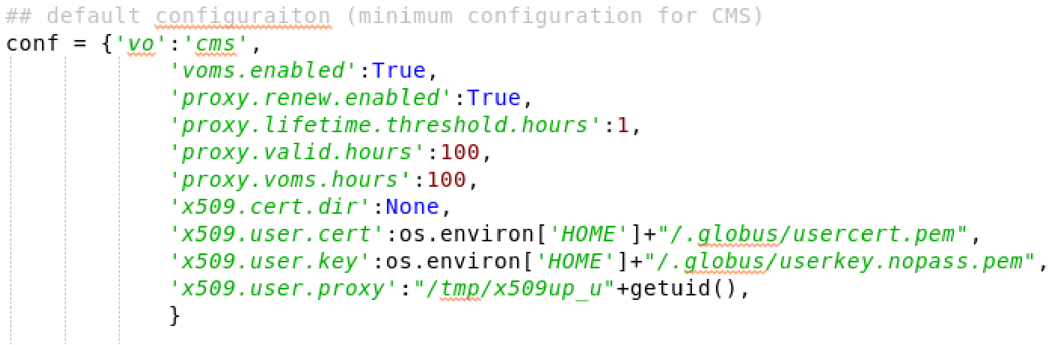
...

Figure 6.24: The HappyFace configuration file structure.

- The *read_happyface_config()* function is used for reading the HappyFace configuration file variable types (int, float, boolean, string, etc). The function is later used by the BaseEnvReader class.

- *BaseEnv* is a base class and is used for keeping class attributes and methods that can be shared later by the inherited GridEnv and the CvmfsEnv class attributes and methods. This is done in order to avoid accidental name conflicts, which may cause hard-to-find bugs in the program.

- *GridEnv* and the *CvmfsEnv* classes are inherited from the BaseEnv class and are used for storing class attributes of the grid and the cvmfs environments correspondingly in case of a problem with the HappyFace configuration file. For example, the GridEnv class overrides the BaseEnv class *conf{}* attribute and looks like the following (Figure 6.25):



```
## default configuraiton (minimum configuration for CMS)
conf = {'vo':'cms',
            'voms.enabled':True,
            'proxy.renew.enabled':True,
            'proxy.lifetime.threshold.hours':1,
            'proxy.valid.hours':100,
            'proxy.voms.hours':100,
            'x509.cert.dir':None,
            'x509.user.cert':os.environ['HOME']+"/.globus/usercert.pem",
            'x509.user.key':os.environ['HOME']+"/.globus/userkey.nopass.pem",
            'x509.user.proxy':"/tmp/x509up_u"+getuid(),
            }
```

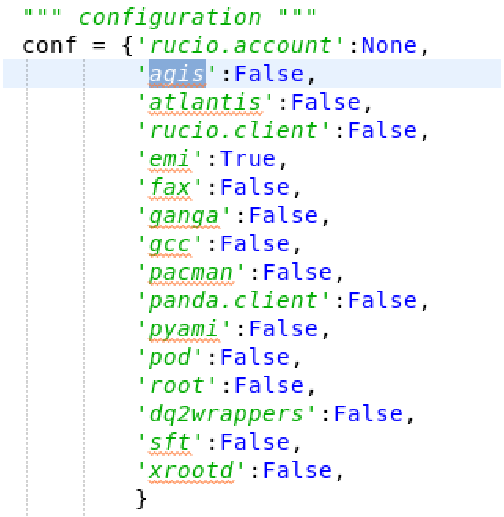
Figure 6.25: The GridEnv class *conf{}* attribute.

The CvmfsEnv class overrides the BaseEnv class *conf{}* attribute and looks like the following (Figure 6.26):

Depending on the class instance, the corresponding *conf{}* attribute will be called.

- *BaseEnvReader* is a base class and is used for setting up the grid or the cvmfs environments by reading and changing the default variables of the HappyFace configuration file.

- The *GridEnvReader* and the *CvmfsEnvReader* classes are inherited from the BaseEnvReader class and are used for setting up the grid and the cvmfs environments correspondingly.



```
""" configuration """
conf = {'rucio.account':None,
        'agis':False,
        'atlantis':False,
        'rucio.client':False,
        'emi':True,
        'fax':False,
        'ganga':False,
        'gcc':False,
        'pacman':False,
        'panda.client':False,
        'pyami':False,
        'pod':False,
        'root':False,
        'dq2wrappers':False,
        'sft':False,
        'xrootd':False,
        }
```

Figure 6.26: The CvmfsEnv *conf{}* class attribute.

- The *gridcertificate.py* Python script consists of the GridCertificate class. The GridCertificate class checks if the X.509 user certificate is valid or not. The *ProxyCertificateHandler* is the inherited class from the GridCertificate class and generates the new proxy certificate in case it is not valid. The ProxyCertificateHandler class has a strong connection to the GridCertificate class and cannot exist without it.

  The UML diagram is displayed in Figure 6.27.

  – The *getUserKey()*, *getUserCert()*, *getUserProxy()* methods get information about the path/location of the X.509 user certificate, key and proxy certificate by using the GridEnvReader class instance.

  – The *checkIfUserCertExists()*, *checkIfUserKeyExists()*, *checkIfGridProxyExists()* methods check an existence of the X.509 user certificate, key and proxy certificate by above mentioned path/location.

  – The *checkIfUserCertHasCorrectOwnership()*, *checkIfUserKeyHasCorrectOwnership()*, *checkIfUserProxyHasCorrectOwnership()* methods check an ownership of the X.509 user certificate, key and proxy certificate files.

  – The *checkIfNoPassphraseUserKeyIs()* method removes the password from the X.509 user key by using the *openssl* command.

  – The *checkIfValidUserKeyExists()* method checks if the X.509 user key is valid by using the *openssl* command.

  – The *checkIfGridProxyIsStillValid()* method checks if the X.509 user proxy is still valid by using the *voms-proxy-info -e* command.

  – The *checkIfGridProxyHasAcTimeleft()* method checks the lifetime of the X.509 proxy certificate by using the *voms-proxy-info -actimeleft*.

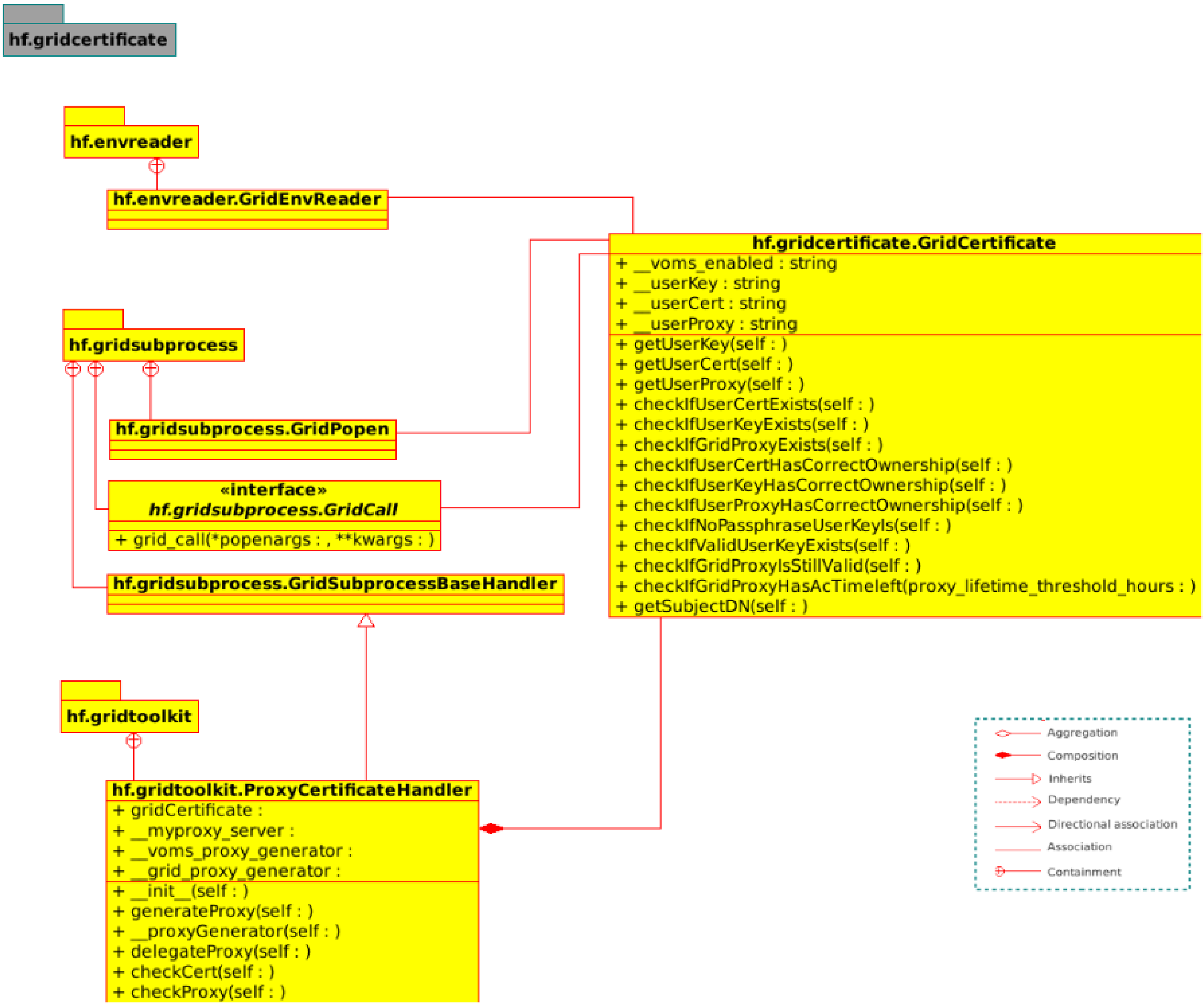

hf.gridcertificate
hf.envreader
hf.envreader.GridEnvReader
hf.gridcertificate.GridCertificate
+ __voms_enabled : string
+ __userKey : string
+ __userCert : string
+ __userProxy : string
+ getUserKey(self :)
+ getUserCert(self :)
+ getUserProxy(self :)
+ checkIfUserCertExists(self :)
+ checkIfUserKeyExists(self :)
+ checkIfGridProxyExists(self :)
+ checkIfUserCertHasCorrectOwnership(self :)
+ checkIfUserKeyHasCorrectOwnership(self :)
+ checkIfUserProxyHasCorrectOwnership(self :)
+ checkIfNoPassphraseUserKeyIs(self :)
+ checkIfValidUserKeyExists(self :)
+ checkIfGridProxyIsStillValid(self :)
+ checkIfGridProxyHasAcTimeleft(proxy_lifetime_threshold_hours :)
+ getSubjectDN(self :)
hf.gridsubprocess
hf.gridsubprocess.GridPopen
«interface»
hf.gridsubprocess.GridCall
+ grid_call(*popenargs : , **kwargs :)
hf.gridsubprocess.GridSubprocessBaseHandler
hf.gridtoolkit
hf.gridtoolkit.ProxyCertificateHandler
+ gridCertificate :
+ __myproxy_server :
+ __voms_proxy_generator :
+ __grid_proxy_generator :
+ __init__(self :)
+ generateProxy(self :)
+ __proxyGenerator(self :)
+ delegateProxy(self :)
+ checkCert(self :)
+ checkProxy(self :)
Aggregation
Composition
Inherits
Dependency
Directional association
Association
Containment

Figure 6.27: The *GridCertificate class* UML diagram.

  - The *getSubjectDN()* method gets the name of owner of the X.509 certificate by using the *openssl* command.

- The *gridsubprocess.py* Python script consists of the GridCalledProcessError, the GridTimeoutExpired, the GridPopen, the GridSubprocessBaseHandler classes and from two functions which are described below. The gridsubprocess.py Python script provides a functionality to execute the grid commands from the HappyFace system. It also exports the X.509 user certificate and the user key and enables the cvmfs environment when executing the grid process. The gridsubprocess.py is an extended module of Python subprocess with added two parameters (grid and cvmfs).

  The UML diagram is displayed in Figure 6.28.

  - The *GridPopen* class inherits from the Python subprocess *Popen* class and is used for opening a subprocess. The *GridPopen* has two more attributes: the *gridSetupLoader* and the *cvmfsSetupLoader* for setting up the grid and the cvmfs environments.

  - The *grid_call()* function is the same Python subprocess *call()* function with the difference that in the function content the *GridPopen* class is called instead of the Python subprocess *Popen* class. The *grid_call()* function waits for the command to complete or timeout, then returns the returncode.

  - The *check_grid_call()* function is the same Python subprocess *check_call()* function with the difference that in the function content, the *grid_call()* function is called instead of the Python subprocess *call()* function. The *check_grid_call()* function waits for the command to complete and if the exit code is zero then returns, otherwise raises the *GridCalledProcessError* exception.

  - The *GridTimeoutExpired* class inherits from the Python *SubprocessError* class. This exception is raised when the timeout expires, while waiting for the child process.

  - The *GridCalledProcessError* class inherits from the Python *Exception* class. This exception is raised when a process run by the *check_call()* returns a non-zero exit status.

  - The *GridSubprocessBaseHandler* class is mainly used for executing the grid commands. It has two methods: *execute()* and *showGridProcess()*. The *execute()* method is just used for executing the grid commands, for example, the *"uberftp -retry 2 -keepalive 10 se-goegrid.gwdg.de 'put /var/tmp/A_random.txt /pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test/' "*. In this example the command copies a file from the local host to the remote host.

    The *showGridProcess()* method shows the current running grid processes. The *GridSubprocessBaseHandler* class is connected with other classes and functions as it is shown in Figure 6.28. Access methods are inherited from the

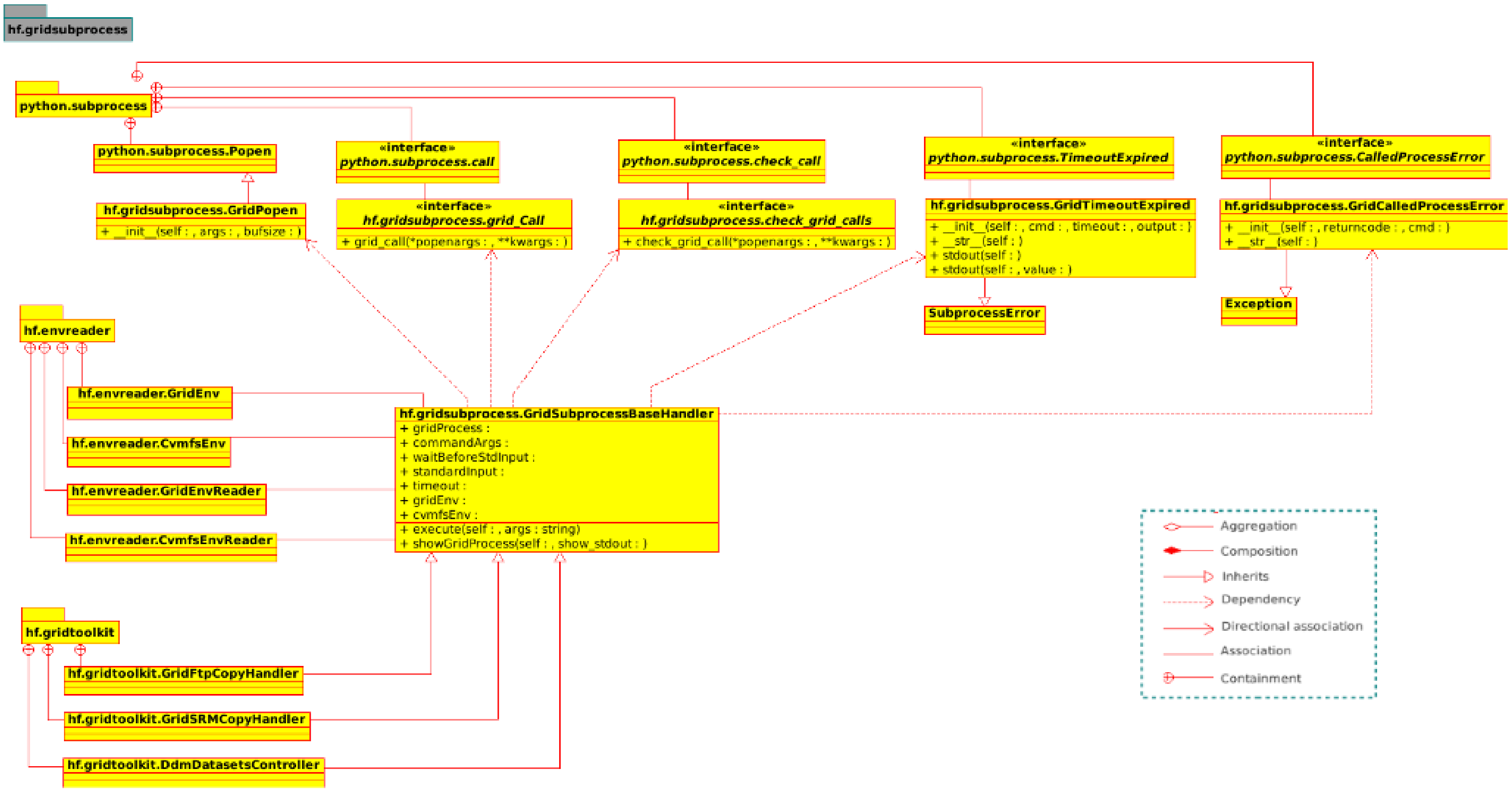

hf.gridsubprocess
python.subprocess
python.subprocess.Popen
hf.gridsubprocess.GridPopen
+ __init__(self : , args : , bufsize :)
«interface»
python.subprocess.call
«interface»
hf.gridsubprocess.grid_Call
+ grid_call(*popenargs : , **kwargs :)
«interface»
python.subprocess.check_call
«interface»
hf.gridsubprocess.check_grid_calls
+ check_grid_call(*popenargs : , **kwargs :)
«interface»
python.subprocess.TimeoutExpired
hf.gridsubprocess.GridTimeoutExpired
+ __init__(self : , cmd : , timeout : , output :)
+ __str__(self :)
+ stdout(self :)
+ stdout(self : , value :)
SubprocessError
«interface»
python.subprocess.CalledProcessError
hf.gridsubprocess.GridCalledProcessError
+ __init__(self : , returncode : , cmd :)
+ __str__(self :)
Exception
hf.envreader
hf.envreader.GridEnv
hf.envreader.CvmfsEnv
hf.envreader.GridEnvReader
hf.envreader.CvmfsEnvReader
hf.gridsubprocess.GridSubprocessBaseHandler
+ gridProcess :
+ commandArgs :
+ waitBeforeStdInput :
+ standardInput :
+ timeout :
+ gridEnv :
+ cvmfsEnv :
+ execute(self : , args : string)
+ showGridProcess(self : , show_stdout :)
hf.gridtoolkit
hf.gridtoolkit.GridFtpCopyHandler
hf.gridtoolkit.GridSRMCopyHandler
hf.gridtoolkit.DdmDatasetsController
Aggregation
Composition
Inherits
Dependency
Directional association
Association
Containment

Figure 6.28: The UML diagram of the *GridSubprocess* class.

GridSubprocessBaseHandler class in order to be able to execute the necessary grid commands.

- Different access methods are implemented such as *GridFTP*, *GridSRM* and *ATLASDDM* classes that are inherited from the *GridSubprocessBaseHandler* class and also have access to the grid system.

  The UML diagram is displayed in Figure 6.29.

  - The *Transfers* class inherits from the *GridSubprocessBaseHandler* class and provides *set()* and *get()* methods for source and destination hosts, ports, transfer types and site names.

  - The *GenerateFile* class randomly generates files in the local file system. The generated files are located at the */tmp* directory. This class also has a method to copy the generated files to the remote grid site.

  - The *SpaceTokens* class provides the *set()* and *get()* methods for commonly used space tokens. There are four: the *scratchdisk*, the *localgroupdisk*, the *proddisk* and the *datadisk*.

  - The *DdmDatasetsController* class inherits from the *GridSubprocessBaseHandler* class and has several methods, such as *whoami()*, *listDatasets()* and *getMetaData()*. These methods provide the Rucio (DDM) (see Section 4.4) command-line from the HappyFace system.

    For example, in the Linux terminal the command *(rucio list-datasets-rse GOE-GRID_LOCALGROUPDISK | sed -n 1,10p)* looks like the following, see Figure 6.30. The above mentioned command lists the first 10 datasets of the *GoeGrid localgroupdisk* space token.

    In HappyFace, the same command *(rucio list-datasets-rse GOEGRID_LOCALGROUPDISK | sed -n 1,50p)* is written in the class method and looks like the following, see Figure 6.31. The number of datasets are taken from the HappyFace *ATLASDDM module* configuration file.

  - The *GridFtpCopyHandler* class inherits from the *GridSubprocessBaseHandler* and the *Transfers* classes. The *GridFtpCopyHandler* class methods provide the *UberFTP* (see Section 6.3.3.1) [139] command-line functionality from the HappyFace system.

    For example, in the Linux terminal the command *(uberftp -retry 2 -keepalive 10 se-goegrid.gwdg.de "mkdir /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test")* looks like the following, see Figure 6.32. The above mentioned command creates the directory named *test*, in the *GoeGrid scratchdisk* space token.

    In HappyFace, the same command *(uberftp -retry 2 -keepalive 10 se-goegrid.gwdg.de "mkdir /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test")* is written in the class method and looks like the following, see Figure 6.33.

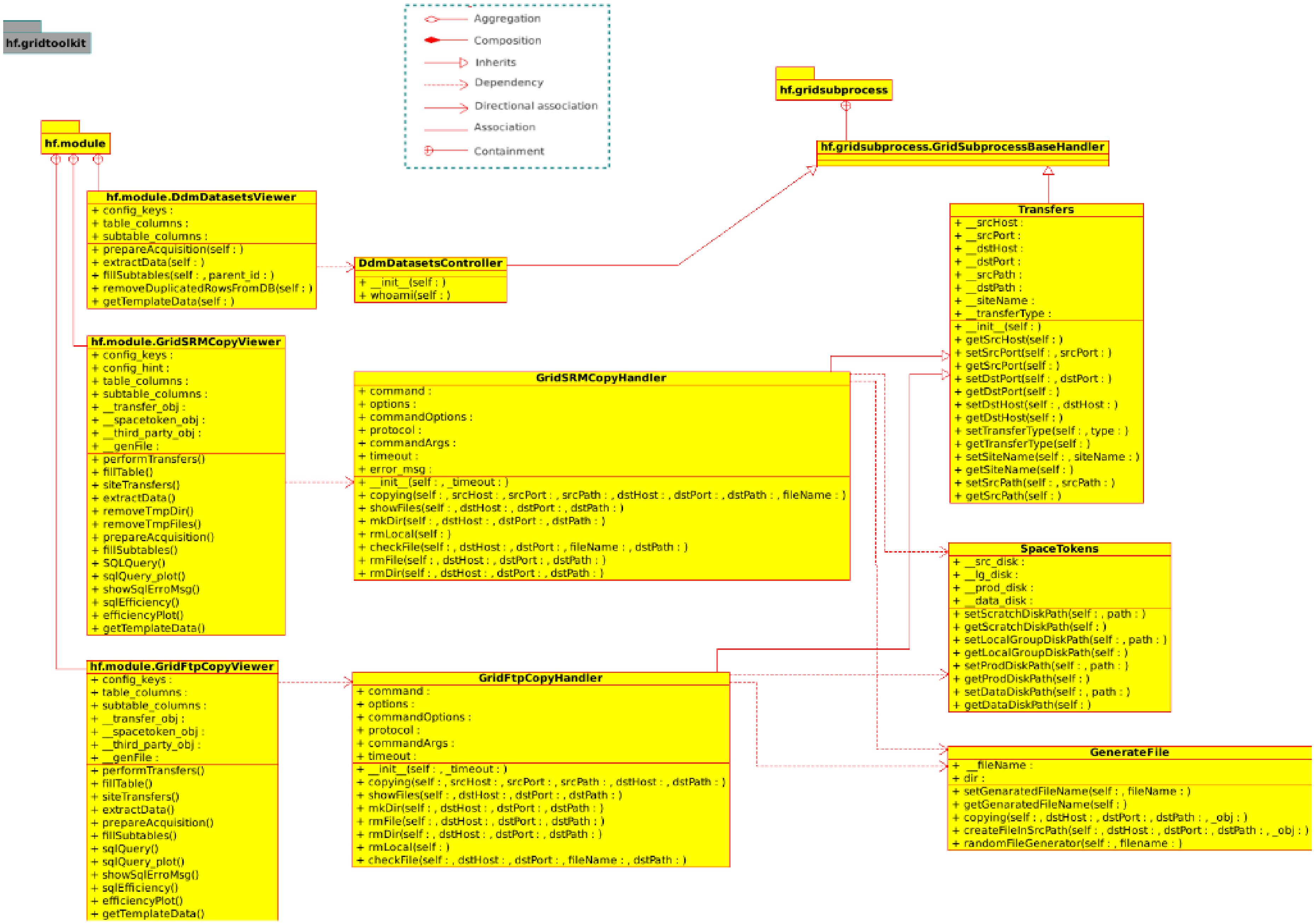
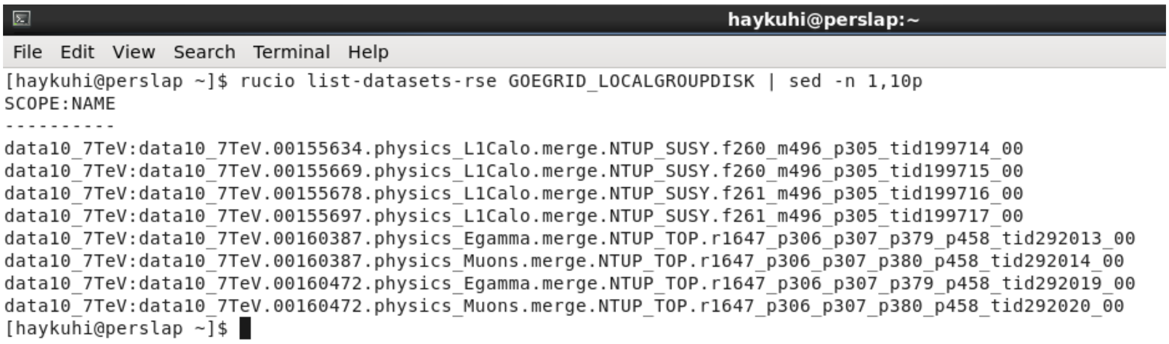
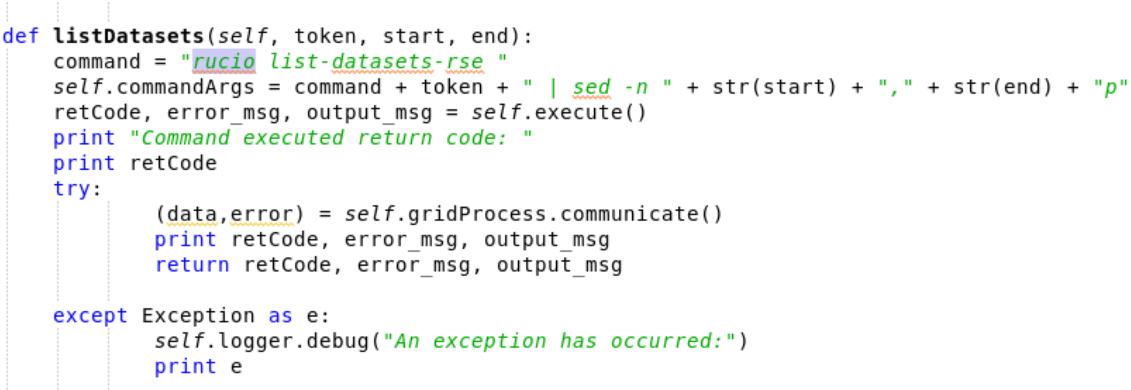

hf.gridtoolkit
Aggregation
Composition
Inherits
Dependency
Directional association
Association
Containment
hf.gridsubprocess
hf.gridsubprocess.GridSubprocessBaseHandler
hf.module
hf.module.DdmDatasetsViewer
+ config_keys :
+ table_columns :
+ subtable_columns :
+ prepareAcquisition(self :)
+ extractData(self :)
+ fillSubtables(self : , parent_id :)
+ removeDuplicatedRowsFromDB(self :)
+ getTemplateData(self :)
DdmDatasetsController
+ __init__(self :)
+ whoami(self :)
Transfers
+ __srcHost :
+ __srcPort :
+ __dstHost :
+ __dstPort :
+ __srcPath :
+ __dstPath :
+ __siteName :
+ __transferType :
+ __init__(self :)
+ getSrcHost(self :)
+ setSrcPort(self : , srcPort :)
+ getSrcPort(self :)
+ setDstPort(self : , dstPort :)
+ getDstPort(self :)
+ setDstHost(self : , dstHost :)
+ getDstHost(self :)
+ setTransferType(self : , type :)
+ getTransferType(self :)
+ setSiteName(self : , siteName :)
+ getSiteName(self :)
+ setSrcPath(self : , srcPath :)
+ getSrcPath(self :)
hf.module.GridSRMCopyViewer
+ config_keys :
+ config_hint :
+ table_columns :
+ subtable_columns :
+ __transfer_obj :
+ __spacetoken_obj :
+ __third_party_obj :
+ __genFile :
+ performTransfers()
+ fillTable()
+ siteTransfers()
+ extractData()
+ removeTmpDir()
+ removeTmpFiles()
+ prepareAcquisition()
+ fillSubtables()
+ SQLQuery()
+ sqlQuery_plot()
+ showSqlErroMsg()
+ sqlEfficiency()
+ efficiencyPlot()
+ getTemplateData()
GridSRMCopyHandler
+ command :
+ options :
+ commandOptions :
+ protocol :
+ commandArgs :
+ timeout :
+ error_msg :
+ __init__(self : , _timeout :)
+ copying(self : , srcHost : , srcPort : , srcPath : , dstHost : , dstPort : , dstPath : , fileName :)
+ showFiles(self : , dstHost : , dstPort : , dstPath :)
+ mkDir(self : , dstHost : , dstPort : , dstPath :)
+ rmLocal(self :)
+ checkFile(self : , dstHost : , dstPort : , fileName : , dstPath :)
+ rmFile(self : , dstHost : , dstPort : , dstPath :)
+ rmDir(self : , dstHost : , dstPort : , dstPath :)
SpaceTokens
+ __src_disk :
+ __lg_disk :
+ __prod_disk :
+ __data_disk :
+ setScratchDiskPath(self : , path :)
+ getScratchDiskPath(self :)
+ setLocalGroupDiskPath(self : , path :)
+ getLocalGroupDiskPath(self :)
+ setProdDiskPath(self : , path :)
+ getProdDiskPath(self :)
+ setDataDiskPath(self : , path :)
+ getDataDiskPath(self :)
hf.module.GridFtpCopyViewer
+ config_keys :
+ table_columns :
+ subtable_columns :
+ __transfer_obj :
+ __spacetoken_obj :
+ __third_party_obj :
+ __genFile :
+ performTransfers()
+ fillTable()
+ siteTransfers()
+ extractData()
+ prepareAcquisition()
+ fillSubtables()
+ sqlQuery()
+ sqlQuery_plot()
+ showSqlErroMsg()
+ sqlEfficiency()
+ efficiencyPlot()
+ getTemplateData()
GridFtpCopyHandler
+ command :
+ options :
+ commandOptions :
+ protocol :
+ commandArgs :
+ timeout :
+ __init__(self : , _timeout :)
+ copying(self : , srcHost : , srcPort : , srcPath : , dstHost : , dstPath :)
+ showFiles(self : , dstHost : , dstPort : , dstPath :)
+ mkDir(self : , dstHost : , dstPort : , dstPath :)
+ rmFile(self : , dstHost : , dstPort : , dstPath :)
+ rmDir(self : , dstHost : , dstPort : , dstPath :)
+ rmLocal(self :)
+ checkFile(self : , dstHost : , dstPort : , fileName : , dstPath :)
GenerateFile
+ __fileName :
+ dir :
+ setGenaratedFileName(self : , fileName :)
+ getGenaratedFileName(self :)
+ copying(self : , dstHost : , dstPort : , dstPath : , _obj :)
+ createFileInSrcPath(self : , dstHost : , dstPort : , dstPath : , _obj :)
+ randomFileGenerator(self : , filename :)

Figure 6.29: The UML diagram of the *GridFTP*, the *GridSRM* and the *ATLASDDM* classes.



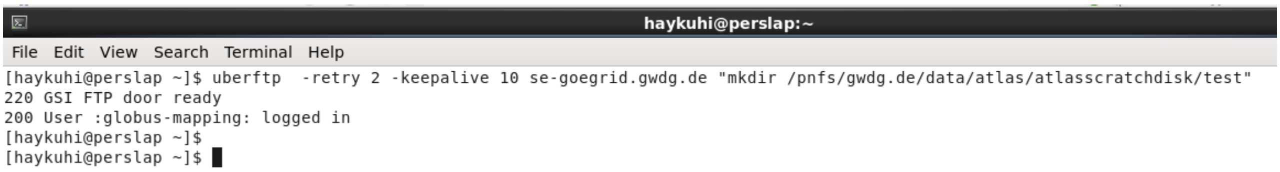
```
[haykuhi@perslap ~]$ rucio list-datasets-rse GOEGRID_LOCALGROUPDISK | sed -n 1,10p
SCOPE:NAME
----------
data10_7TeV:data10_7TeV.00155634.physics_L1Calo.merge.NTUP_SUSY.f260_m496_p305_tid199714_00
data10_7TeV:data10_7TeV.00155669.physics_L1Calo.merge.NTUP_SUSY.f260_m496_p305_tid199715_00
data10_7TeV:data10_7TeV.00155678.physics_L1Calo.merge.NTUP_SUSY.f261_m496_p305_tid199716_00
data10_7TeV:data10_7TeV.00155697.physics_L1Calo.merge.NTUP_SUSY.f261_m496_p305_tid199717_00
data10_7TeV:data10_7TeV.00160387.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292013_00
data10_7TeV:data10_7TeV.00160387.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292014_00
data10_7TeV:data10_7TeV.00160472.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292019_00
data10_7TeV:data10_7TeV.00160472.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292020_00
[haykuhi@perslap ~]$ █
```

Figure 6.30: Running the *(rucio list-datasets-rse GOEGRID_LOCALGROUPDISK | sed -n 1,10p)* command in the Linux terminal.



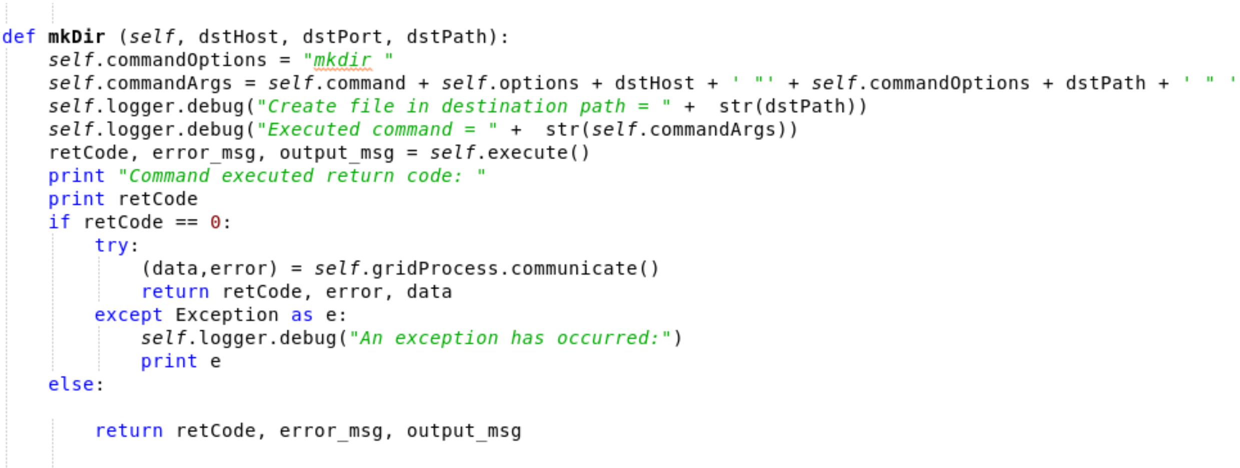
```
def listDatasets(self, token, start, end):
    command = "rucio list-datasets-rse "
    self.commandArgs = command + token + " | sed -n " + str(start) + "," + str(end) + "p"
    retCode, error_msg, output_msg = self.execute()
    print "Command executed return code: "
    print retCode
    try:
            (data,error) = self.gridProcess.communicate()
            print retCode, error_msg, output_msg
            return retCode, error_msg, output_msg

    except Exception as e:
            self.logger.debug("An exception has occurred:")
            print e
```

Figure 6.31: Running the *(rucio list-datasets-rse GOEGRID_LOCALGROUPDISK | sed -n 1,50p)* command from the HappyFace system. The value of *str(start)* and *str(end)* variables are taken from the *ATLASDDM module* configuration file.



```
[haykuhi@perslap ~]$ uberftp  -retry 2 -keepalive 10 se-goegrid.gwdg.de "mkdir /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test"
220 GSI FTP door ready
200 User :globus-mapping: logged in
[haykuhi@perslap ~]$
[haykuhi@perslap ~]$ █
```

Figure 6.32: Running the *(uberftp -retry 2 -keepalive 10 se-goegrid.gwdg.de "mkdir /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test")* command in the Linux terminal.



```
def mkDir (self, dstHost, dstPort, dstPath):
    self.commandOptions = "mkdir "
    self.commandArgs = self.command + self.options + dstHost + ' "' + self.commandOptions + dstPath + ' " '
    self.logger.debug("Create file in destination path = " +  str(dstPath))
    self.logger.debug("Executed command = " +  str(self.commandArgs))
    retCode, error_msg, output_msg = self.execute()
    print "Command executed return code: "
    print retCode
    if retCode == 0:
        try:
            (data,error) = self.gridProcess.communicate()
            return retCode, error, data
        except Exception as e:
            self.logger.debug("An exception has occurred:")
            print e
    else:

        return retCode, error_msg, output_msg
```

Figure 6.33: Running the same *(uberftp -retry 2 -keepalive 10 se-goegrid.gwdg.de "mkdir /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test")* command from the HappyFace system.

- The *GridSRMCopyHandler* class inherits from the *GridSubprocessBaseHandler* and the *Transfers* classes. The *GridSRMCopyHandler* class methods provide the *SRM*(see Section 6.3.3.2) [140] command-line functionality from the HappyFace system. The *GridSRMCopyHandler* class has similar functionality as the *GridFtpCopyHandler* class but with a different command-line.

The new grid-enabled HappyFace extension was successfully integrated and it provides direct access to the grid infrastructure.

Three main grid-enabled modules, the so-called GridFTP, the GridSRM and the ATLASDDM modules were implemented.

The GridFTP and the GridSRM modules are designed to show the efficiency of the particular grid site and to check the performance of the grid transfers among the site space tokens and other grid sites. They check the status of transfers between site space tokens by using two different command-lines (UberFTP and SRM). The GridFTP and the GridSRM modules have a transfer efficiency plot that shows the number of successful and failed transfers for the site space tokens for the given time period. The existence of the modules allow system administrators to know whether the transfers between site space tokens succeed or failed and in case of failure points out the reason of the failure.

The ATLASDDM module shows all current datasets that exist in the given storage disk or the storage pool. This allows system administrators to see all different datasets that are stored in the storage systems.

6.3.3 Results

6.3.3.1 GridFTP Module

The *UberFTP* [64], [139] is a gridFTP-enabled client that supports both interactive use and the FTP commands on the uberftp command-line to transfer files between two computers. The UberFTP is intended for use with computers that have a GridFTP server installed. It supports GSI authentication, parallel data channels and striping.

The plot in the web interface shows the transfer efficiency of the GoeGrid site space tokens for the given time period by using UberFTP commands.

In GoeGrid, there are four main space tokens that need to be monitored. They are: the scratchdisk, the localgroupdisk, the proddisk and the datadisk. The space token transfer efficiency is defined by the number of successful and failed transfers, which are stored in the HappyFace database. In the plot, the green bar shows the number of successful transfers and the red bar shows the number of failed transfers for the particular space token.

For example, in this particular case, there are 57 successful and 30 failed transfers for the GoeGrid localgroupdisk space token (Figure 6.34).

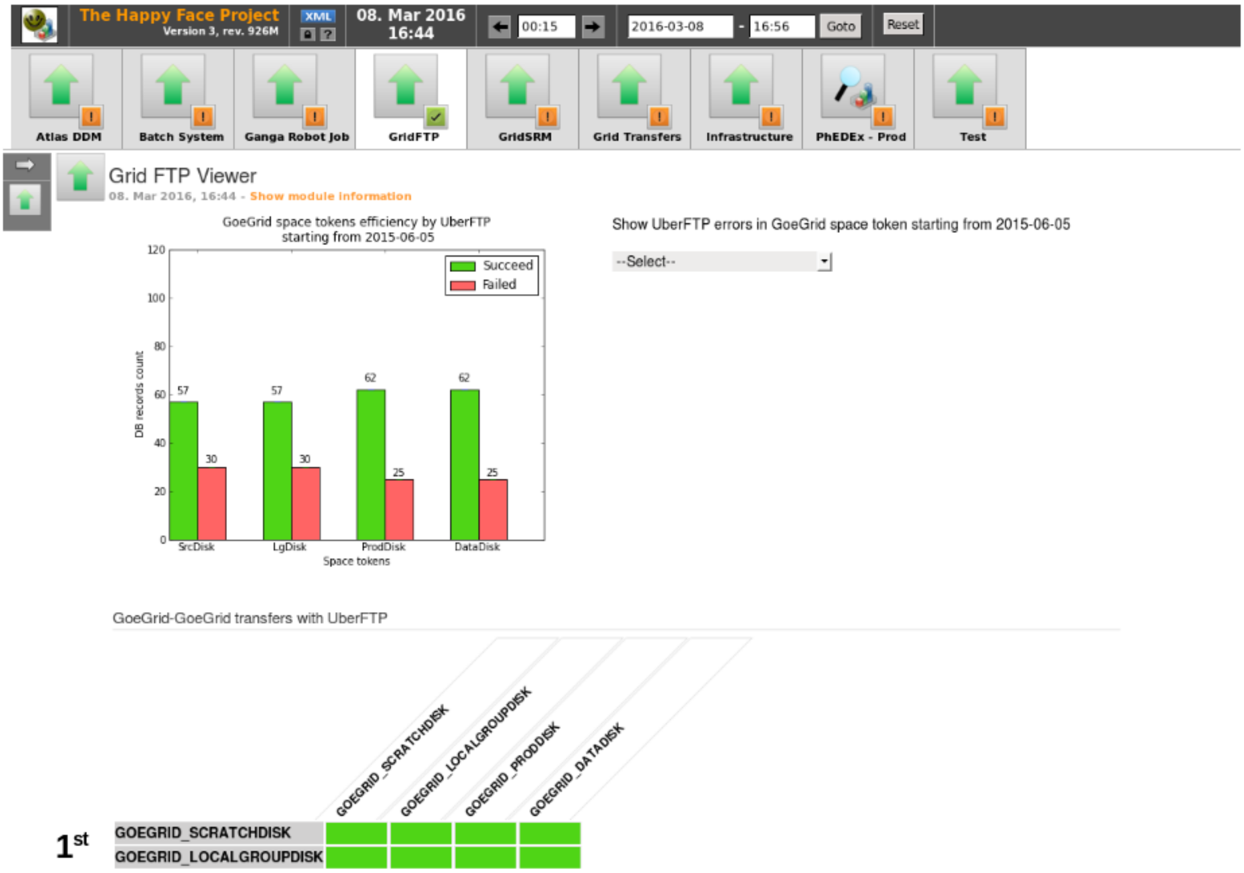
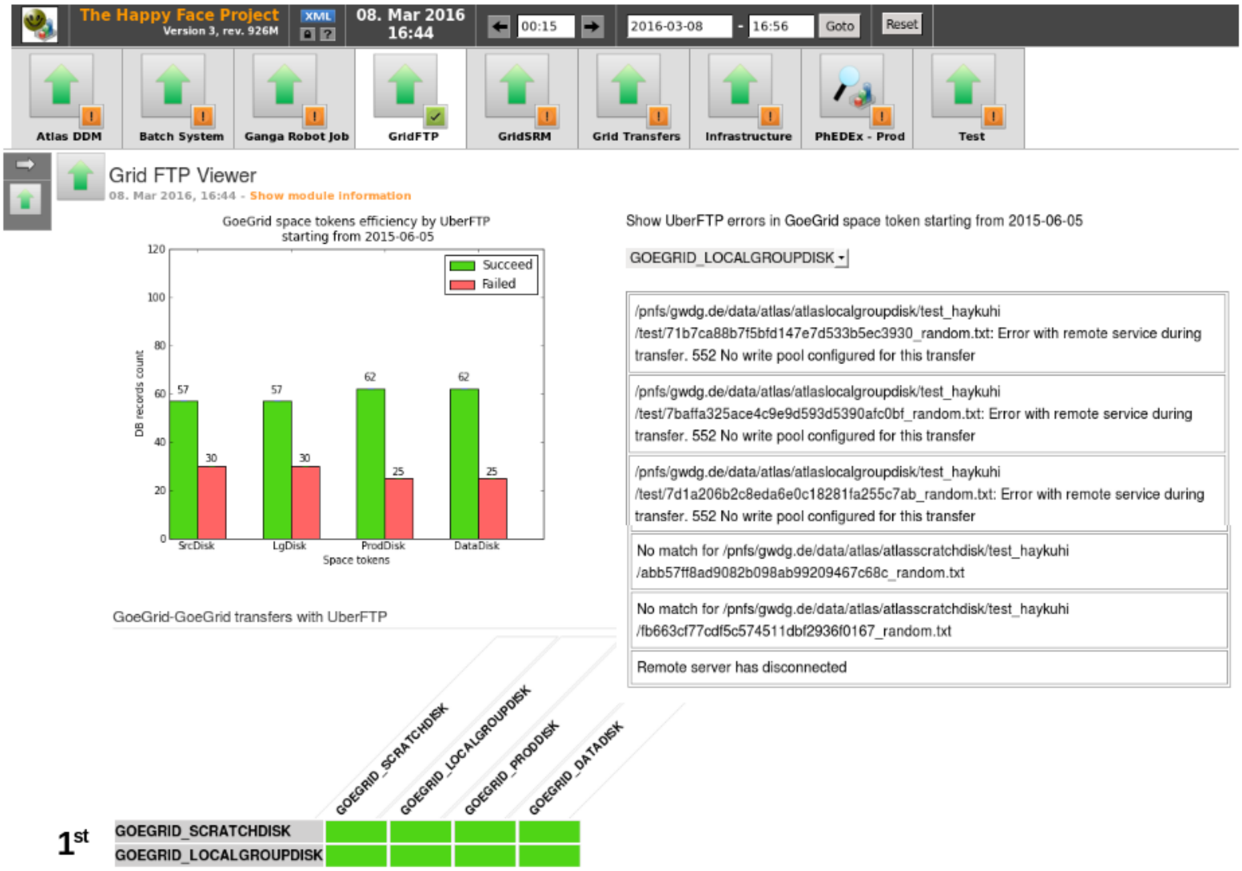


Figure 6.34: The GridFTP module web output.

The combobox, which is on the right side of the plot shows the existing failures in the selected space token for the given time period. By choosing a space token (for example, localgroupdisk) more detailed information about the failures on the space token will be listed in the table (Figure 6.35). In this case, the existing 30 failures are due to the following error messages:

- error with remote service during transfer,
- problem with the file, which does not exists,
- remote server is disconnected.

The extraction of space token failure information is done by using SQLite query with similar syntax *"SELECT DISTINCT ... FROM ... WHERE ...;"*. The SQLite DISTINCT clause removes duplicated rows from the result set.



Show UberFTP errors in GoeGrid space token starting from 2015-06-05

GOEGRID_LOCALGROUPDISK

/pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test_haykuhi /test/71b7ca88b7f5bfd147e7d533b5ec3930_random.txt: Error with remote service during transfer. 552 No write pool configured for this transfer

/pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test_haykuhi /test/7baffa325ace4c9e9d593d5390afc0bf_random.txt: Error with remote service during transfer. 552 No write pool configured for this transfer

/pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test_haykuhi /test/7d1a206b2c8eda6e0c18281fa255c7ab_random.txt: Error with remote service during transfer. 552 No write pool configured for this transfer

No match for /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test_haykuhi /abb57ff8ad9082b098ab99209467c68c_random.txt

No match for /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test_haykuhi /fb663cf77cdf5c574511dbf2936f0167_random.txt

Remote server has disconnected

	GOEGRID_SCRATCHDISK	GOEGRID_LOCALGROUPDISK	GOEGRID_PRODDISK	GOEGRID_DATADISK
GOEGRID_SCRATCHDISK	🟩	🟩	🟩	🟩
GOEGRID_LOCALGROUPDISK	🟩	🟩	🟩	🟩

Figure 6.35: The GridFTP module web output.

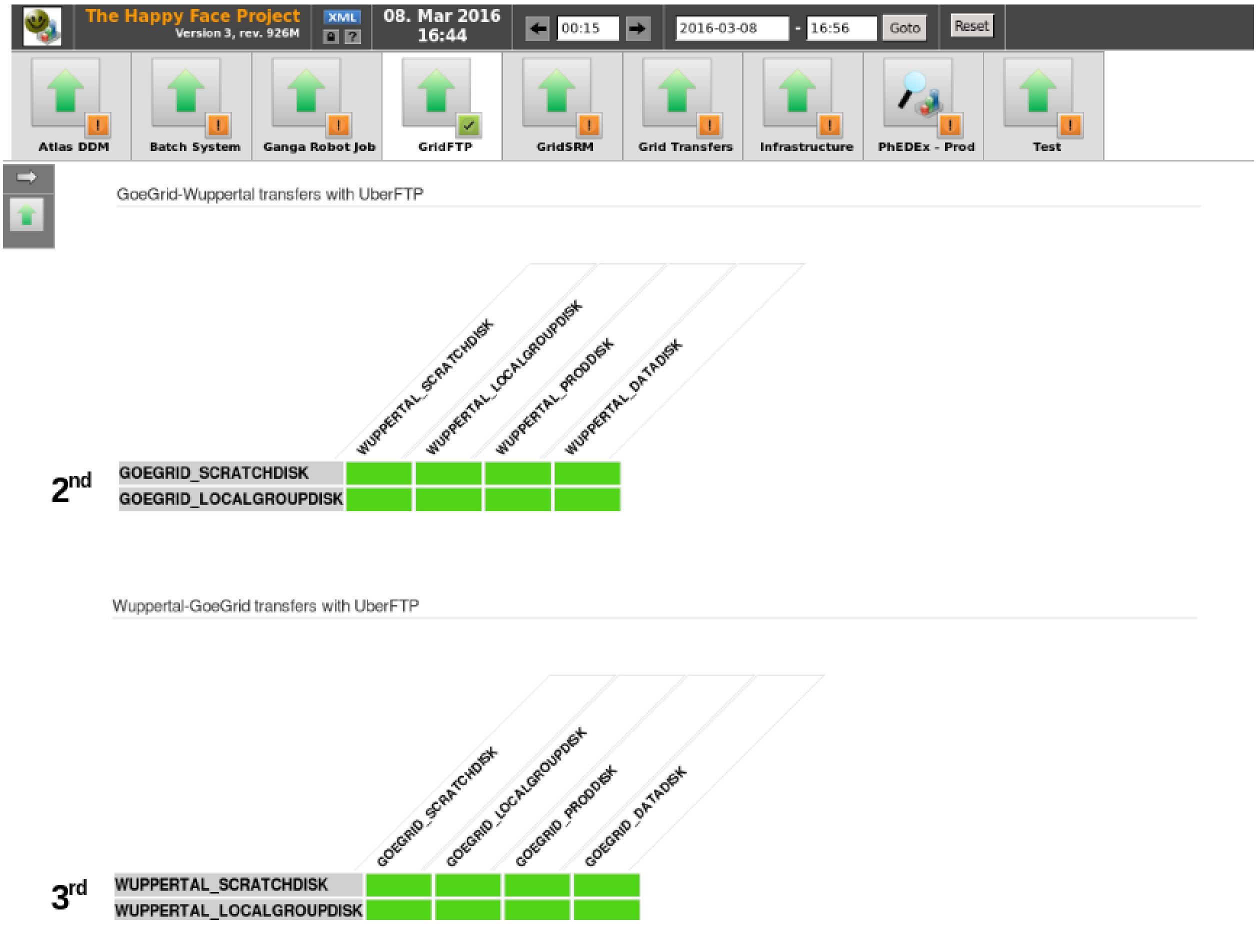
The transfers themselves are shown in three different tables on the HappyFace web interface (Figure 6.36). The 1st table shows transfers from GoeGrid to GoeGrid by using UberFTP commands. The 2nd table shows transfers from GoeGrid to Wuppertal (another Tier-2 site) by using UberFTP commands. The 3rd table shows transfers from Wuppertal to GoeGrid by using UberFTP commands.

The green color indicates successful transfers and the red color with tooltip icon indicates the failures. By moving the cursor on the icon the tooltip window will appear with the particular error message.

The program randomly generates files and copies them from one space token to any other. After overviewing of the transfers it removes all created files and folders.

6.3.3.2 GridSRM Module

The Storage Resource Manager (SRM) [97], [140] is the grid middleware component that provides a dynamic space allocation, file management on shared storage resources in the grid. Different storage system implementations (dCache, StoRM) are based on the same SRM specification, see 2.1.5.3.



GoeGrid-Wuppertal transfers with UberFTP

	WUPPERTAL_SCRATCHDISK	WUPPERTAL_LOCALGROUPDISK	WUPPERTAL_PRODDISK	WUPPERTAL_DATADISK
GOEGRID_SCRATCHDISK	🟩	🟩	🟩	🟩
GOEGRID_LOCALGROUPDISK	🟩	🟩	🟩	🟩

Wuppertal-GoeGrid transfers with UberFTP

	GOEGRID_SCRATCHDISK	GOEGRID_LOCALGROUPDISK	GOEGRID_PRODDISK	GOEGRID_DATADISK
WUPPERTAL_SCRATCHDISK	🟩	🟩	🟩	🟩
WUPPERTAL_LOCALGROUPDISK	🟩	🟩	🟩	🟩

Figure 6.36: GridFTP module web output.

The plot in the web interface shows the transfer efficiency of the GoeGrid site space tokens for a given time period by using SRM commands. In case of the GoeGrid, there are four main space tokens that need to be monitored. They are: scratchdisk, localgroupdisk, proddisk and datadisk. The space token transfer efficiency is defined by the number of successful and failed transfers that are stored in the HappyFace database.

As described in 6.3.3.1 also here, the plot (Figure 6.37) shows the number of successful and failed transfers for the particular space token.

For example, in this particular case, there are 18 successful and 18 failed transfers for the GoeGrid scratchdisk space token.

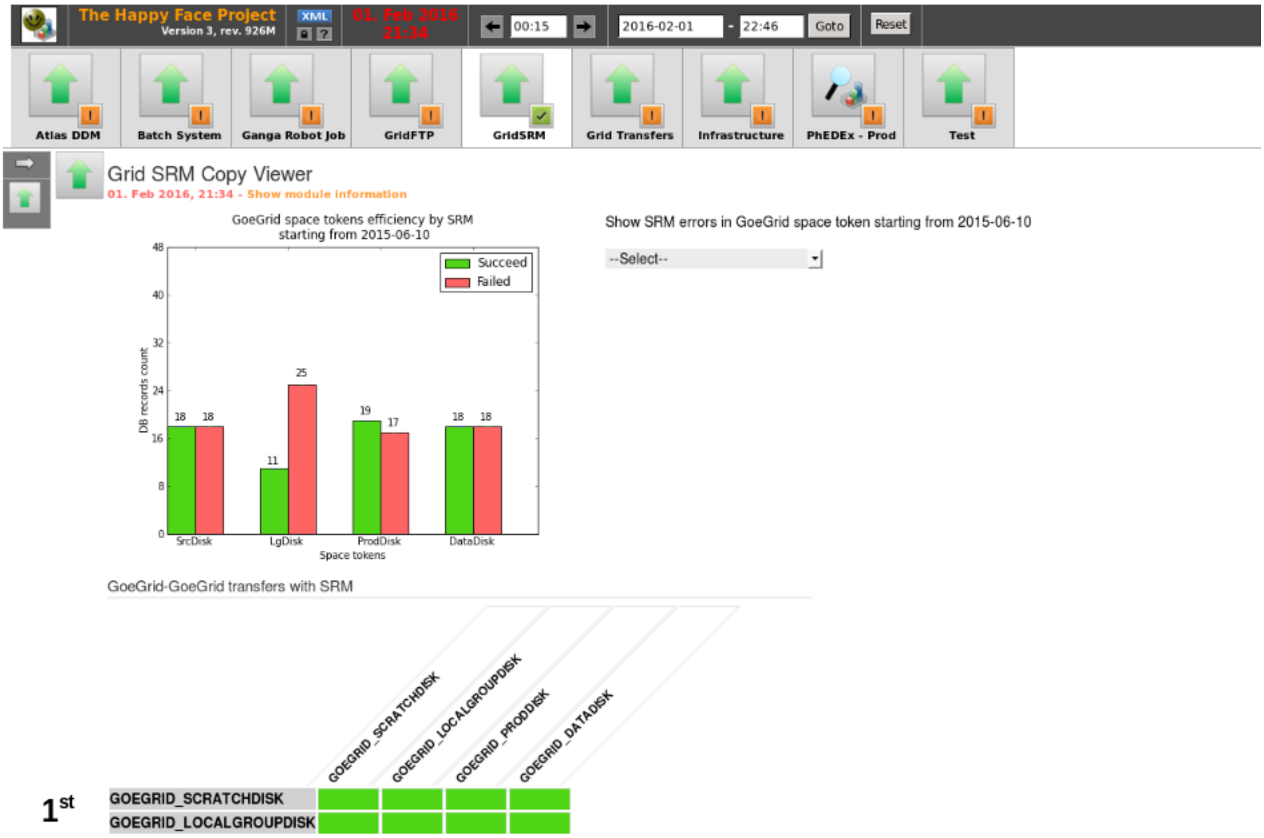
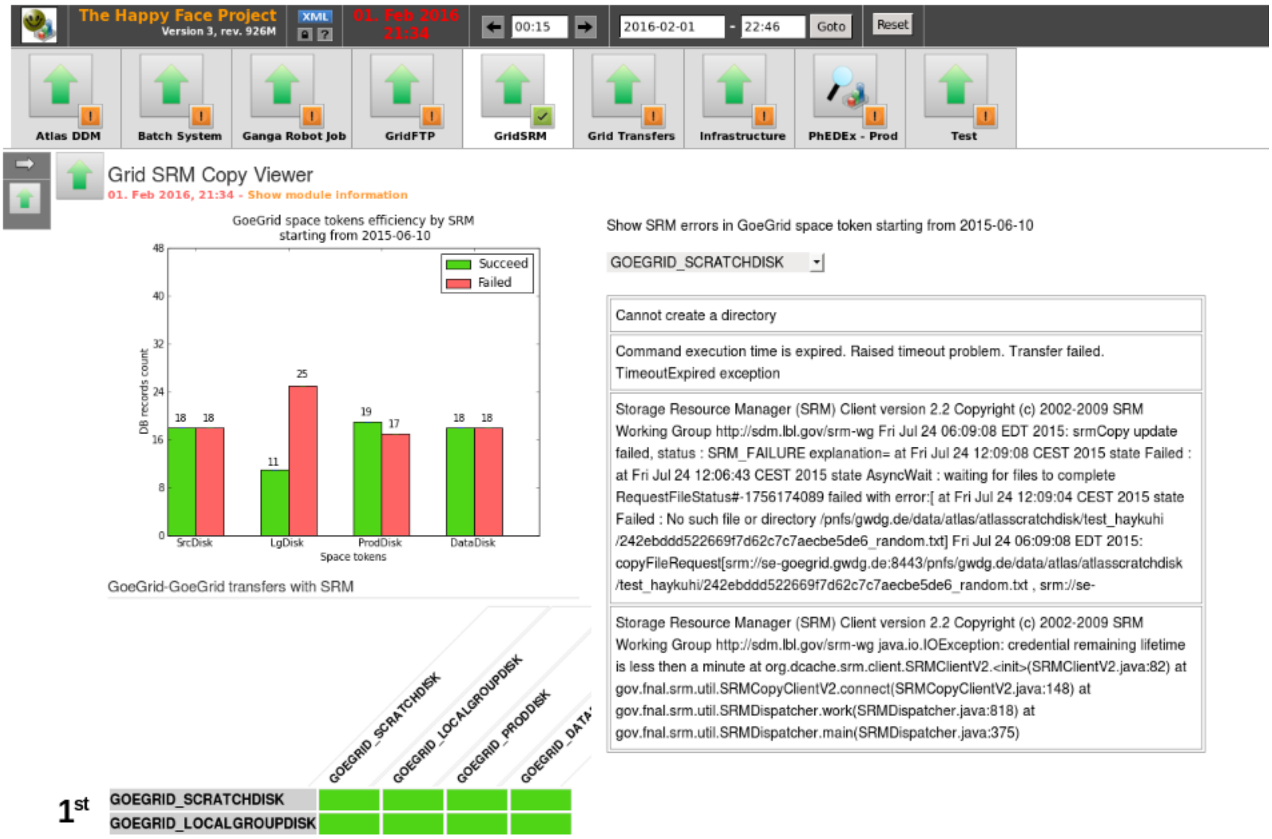


Figure 6.37: GridSRM module web output.

The combobox which is on the right side of the plot shows existing failures in the selected space token for the given time period. By choosing a space token (for example, scratchdisk) more detailed information about the failures on the space token will be listed on the table (Figure 6.38). In this case, the existing 18 failures are due to the following error messages:

- failure of creating the directory,
- command execution timeout,
- problem with the file or directory, which does not exist.

The extraction of a space token failure information is done by using SQLite query described in 6.3.3.1.



	GOEGRID_SCRATCHDISK	GOEGRID_LOCALGROUPDISK	GOEGRID_PRODDISK	GOEGRID_DATA
GOEGRID_SCRATCHDISK	🟩	🟩	🟩	🟩
GOEGRID_LOCALGROUPDISK	🟩	🟩	🟩	🟩

Show SRM errors in GoeGrid space token starting from 2015-06-10

GOEGRID_SCRATCHDISK

```
Cannot create a directory
```

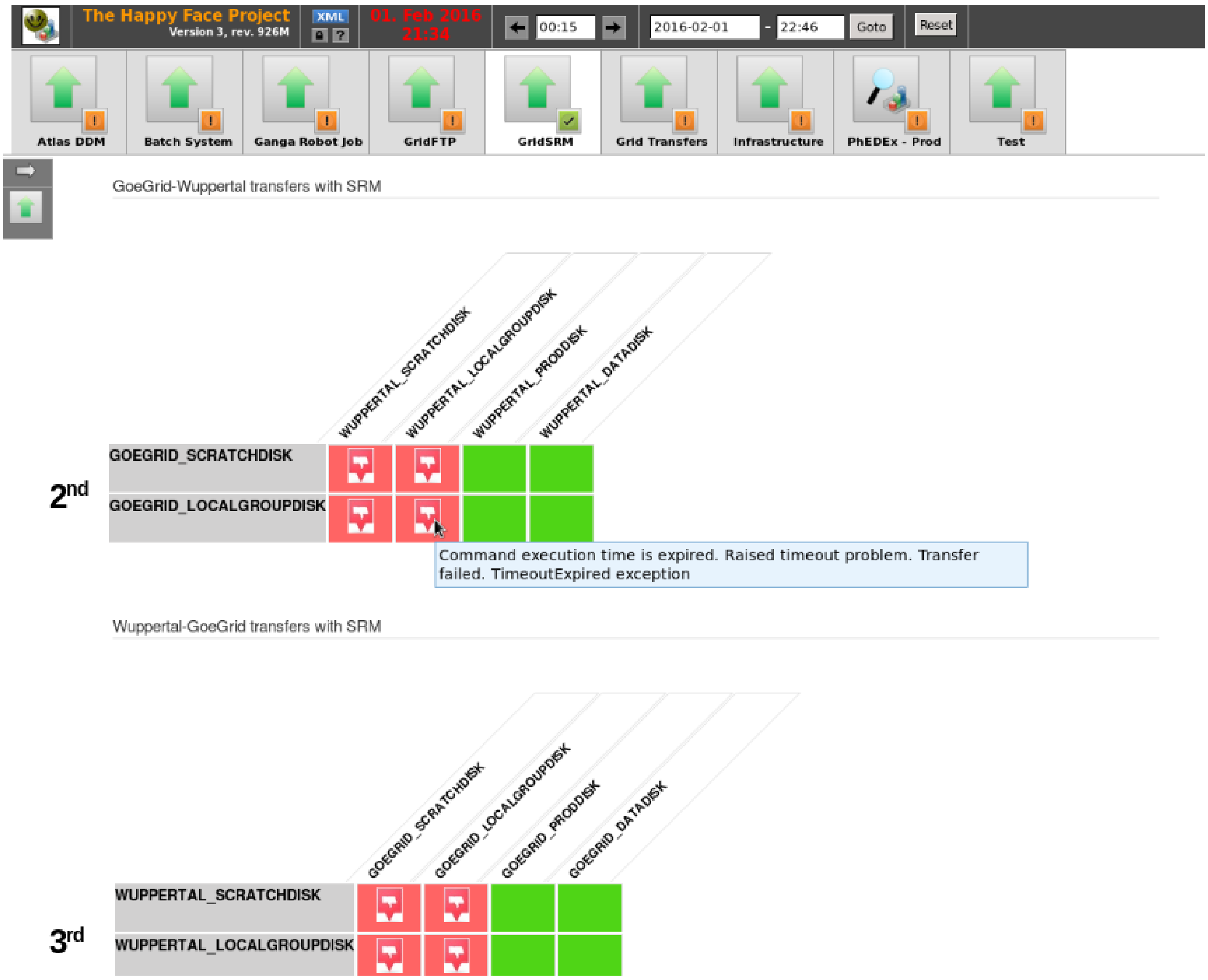
```
Command execution time is expired. Raised timeout problem. Transfer failed.
TimeoutExpired exception
```

```
Storage Resource Manager (SRM) Client version 2.2 Copyright (c) 2002-2009 SRM
Working Group http://sdm.lbl.gov/srm-wg Fri Jul 24 06:09:08 EDT 2015: srmCopy update
failed, status : SRM_FAILURE explanation= at Fri Jul 24 12:09:08 CEST 2015 state Failed :
at Fri Jul 24 12:06:43 CEST 2015 state AsyncWait : waiting for files to complete
RequestFileStatus#-1756174089 failed with error:[ at Fri Jul 24 12:09:04 CEST 2015 state
Failed : No such file or directory /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test_haykuhi
/242ebddd522669f7d62c7c7aecbe5de6_random.txt] Fri Jul 24 06:09:08 EDT 2015:
copyFileRequest[srm://se-goegrid.gwdg.de:8443/pnfs/gwdg.de/data/atlas/atlasscratchdisk
/test_haykuhi/242ebddd522669f7d62c7c7aecbe5de6_random.txt , srm://se-
```

```
Storage Resource Manager (SRM) Client version 2.2 Copyright (c) 2002-2009 SRM
Working Group http://sdm.lbl.gov/srm-wg java.io.IOException: credential remaining lifetime
is less then a minute at org.dcache.srm.client.SRMClientV2.<init>(SRMClientV2.java:82) at
gov.fnal.srm.util.SRMCopyClientV2.connect(SRMCopyClientV2.java:148) at
gov.fnal.srm.util.SRMDispatcher.work(SRMDispatcher.java:818) at
gov.fnal.srm.util.SRMDispatcher.main(SRMDispatcher.java:375)
```

Figure 6.38: The GridSRM module web output.

As described in 6.3.3.1 also here, the transfers themselves are shown in three different tables on the HappyFace web interface (Figure 6.39). The 1st table shows from GoeGrid to GoeGrid transfers by using SRM commands. The 2nd table shows transfers from GoeGrid to Wuppertal (another site) by using SRM commands. The 3rd table shows transfers from Wuppertal to GoeGrid by using SRM commands.



GoeGrid-Wuppertal transfers with SRM

	WUPPERTAL_SCRATCHDISK	WUPPERTAL_LOCALGROUPDISK	WUPPERTAL_PRODDISK	WUPPERTAL_DATADISK
GOEGRID_SCRATCHDISK	🟥	🟥	🟩	🟩
GOEGRID_LOCALGROUPDISK	🟥	🟥	🟩	🟩

Command execution time is expired. Raised timeout problem. Transfer failed. TimeoutExpired exception

Wuppertal-GoeGrid transfers with SRM

	GOEGRID_SCRATCHDISK	GOEGRID_LOCALGROUPDISK	GOEGRID_PRODDISK	GOEGRID_DATADISK
WUPPERTAL_SCRATCHDISK	🟥	🟥	🟩	🟩
WUPPERTAL_LOCALGROUPDISK	🟥	🟥	🟩	🟩

Figure 6.39: The GridSRM module web output.

The tooltip window indicates the particular error message as it is shown in Figure 6.39.

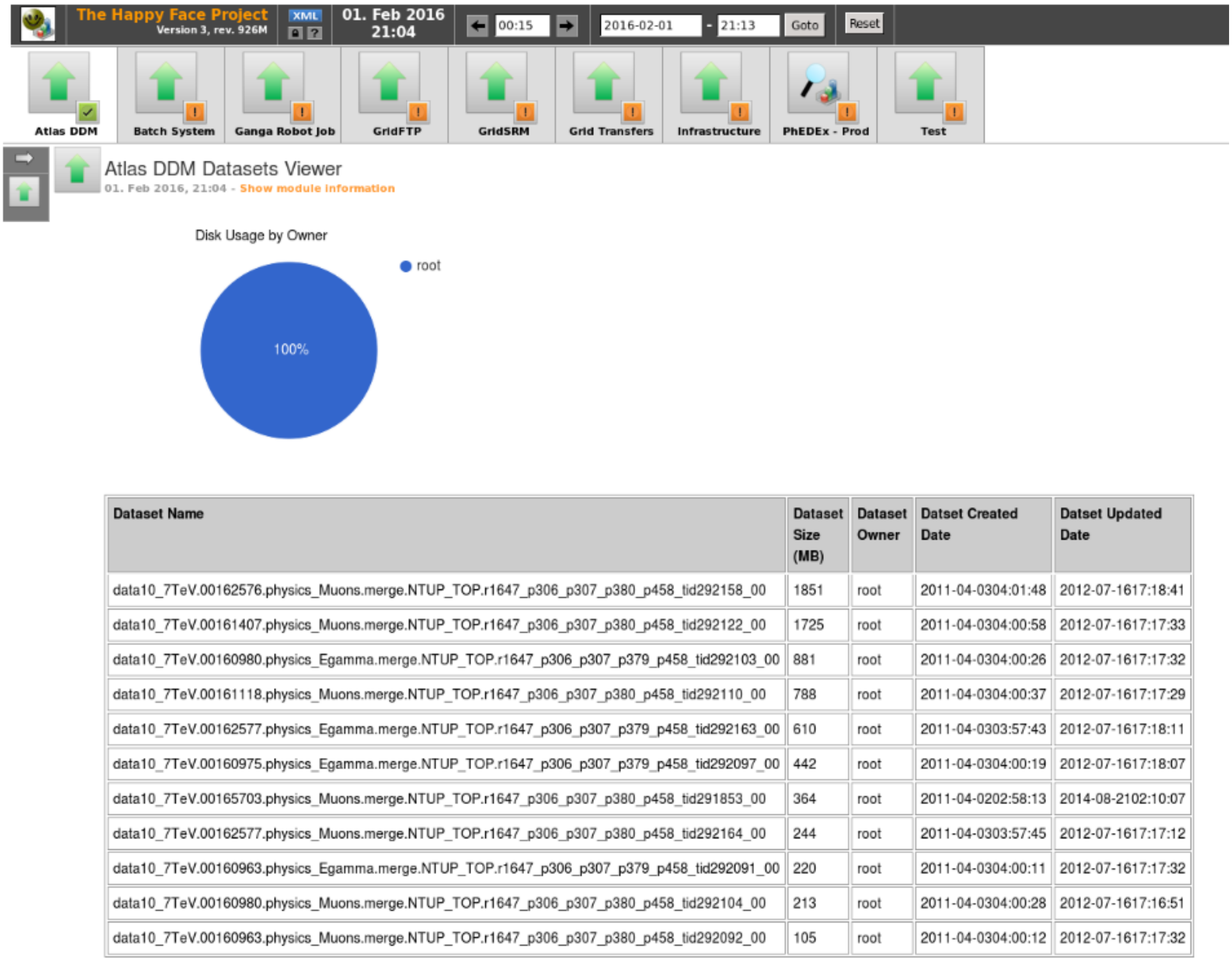
After performing the transfers, all files copied by the program are removed from the space tokens.

The main difference between GridFTP and GridSRM modules is the used command-line. The GridFTP module uses the UberFTP command-line, while the GridSRM module uses the SRM command-line.

6.3.3.3 ATLASDDM Module

The ATLAS DDM module shows all current datasets that exist on the given storage disk or the storage pool. The ATLAS DDM module displays the name of the dataset as well as the size, the owner, the date of creation and the last date of update. On the web interface the table is ordered by descending dataset size and has a limited number of rows (defined in the config file). System administrators can then use the ATLAS DDM module information to manage the storage area by optimising the use of disk space based on any of the displayed factors.

On the web interface of the module, a pie chart displays a random sampling of datasets (Figure 6.40). This gives the system administrator an idea of how much space a particular user consumes.



Dataset Name	Dataset Size (MB)	Dataset Owner	Datset Created Date	Datset Updated Date
data10_7TeV.00162576.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292158_00	1851	root	2011-04-0304:01:48	2012-07-1617:18:41
data10_7TeV.00161407.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292122_00	1725	root	2011-04-0304:00:58	2012-07-1617:17:33
data10_7TeV.00160980.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292103_00	881	root	2011-04-0304:00:26	2012-07-1617:17:32
data10_7TeV.00161118.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292110_00	788	root	2011-04-0304:00:37	2012-07-1617:17:29
data10_7TeV.00162577.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292163_00	610	root	2011-04-0303:57:43	2012-07-1617:18:11
data10_7TeV.00160975.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292097_00	442	root	2011-04-0304:00:19	2012-07-1617:18:07
data10_7TeV.00165703.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid291853_00	364	root	2011-04-0202:58:13	2014-08-2102:10:07
data10_7TeV.00162577.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292164_00	244	root	2011-04-0303:57:45	2012-07-1617:17:12
data10_7TeV.00160963.physics_Egamma.merge.NTUP_TOP.r1647_p306_p307_p379_p458_tid292091_00	220	root	2011-04-0304:00:11	2012-07-1617:17:32
data10_7TeV.00160980.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292104_00	213	root	2011-04-0304:00:28	2012-07-1617:16:51
data10_7TeV.00160963.physics_Muons.merge.NTUP_TOP.r1647_p306_p307_p380_p458_tid292092_00	105	root	2011-04-0304:00:12	2012-07-1617:17:32

Figure 6.40: The ATLASDDM module web output.

At present, the HappyFace system aggregates, processes and stores information of both the monitoring resources as well as the direct access to the grid user applications and grid collective services in the WLCG computing systems.

Chapter 7

Conclusions

7.1 Conclusion

This work summarises the recent activity of the HappyFace system and its goal. The HappyFace system provides a summary of all important site specific information available on a single web interface with the possibility to store all previous site statuses (history functionality). It has a modular structure that makes HappyFace easy understandable. This modularity also allows to provide new modules easily and to share them among different sites.

During the years of work on my PhD, HappyFace was making significant progress. A number of various modules, ATLAS and non-ATLAS modules, were designed and developed by the Göttingen group members.

The expected data growth during the LHC Run 2 phase brought certain adjustments to the existing WLCG computing infrastructure. The main software tools have been improved and developed in order to handle large amounts of data. All main software of the ATLAS computing model that were described in chapter 4 had been improved and until now they are under development. Although, the development of the ATLAS computing infrastructure has top priority, it also affects the work of meta-monitoring tools as they are directly dependent on the infrastructure. HappyFace as a meta-monitoring tool should always be attentive to any kind of changes in the ATLAS computing model, as it is directly influencing the modules' work in HappyFace. Therefore, HappyFace modules always need to be adjusted and in particular cases even to be fully rewritten.

Instability of the ATLAS computing model brought the Göttingen group to the conclusion to modify the initial model of HappyFace in a way, that HappyFace is able to provide access to the grid system directly. This is a very crucial aspect, as it solves the problem of being dependent on the monitoring resources that are always under development. Besides this, it provides an advantage to design and develop a wide range of various modules. For example, the developed grid-enabled modules (GridFTP, GridSRM), which are designed to show the efficiency of a particular grid site and to check the performance of the grid transfers among the site space tokens and other grid sites. The ATLAS DDM module shows all current datasets that exist in a given storage disk or a storage pool. These modules are the first step to take HappyFace to a new phase of development.

At present, the HappyFace system aggregates, processes and stores information of both the monitoring resources as well as the direct access to the grid user applications and grid collective services in the WLCG computing systems. In fact, HappyFace can be adapted to any site requirements within WLCG.

Currently, several German grid sites use the HappyFace system for site monitoring:

- Georg-August-University of Göttingen

- Karlsruhe Institute of Technology
- RWTH Aachen University or Rheinisch-Westfälische Technische Hochschule Aachen University

The installation and set up of HappyFace was simplified. In the past, it was only possible to do it manually, which was time consuming and requiring a certain effort. The provided RPM packages simplified and sped up the installation and set up process of HappyFace. Within several minutes, the whole HappyFace system can be installed.

Besides the progress that HappyFace had within the last several years, the grid-enabled extension is opening new features for the future development. A number of different modules can be still designed and implemented.

Appendix A

Agile Software Development

Agile software development [125] is a software development style which is based on some defined principles. It involves adjustable planning, progressive development, early release, and continuous improvement. Agile software development focuses on keeping code simple, testing frequently and delivering functional bits of the application as soon as they are ready.

The agile software development is based on the following main principles:

- Satisfy customers on early and continuous release by developed software
- Perform changes at any time, even on the final stage of the software development
- Make a release of the software quite often, for example every 2 weeks
- Constant communication between customers and software developers. Face-to-face communication is the best form of communication
- In some stage of the software development, working software is considered to be a progress
- Constant development is able to provide a constant speed
- Continuous attention to the source code development style
- Adaptive way of working

From above mentioned principles, the advantages of agile software development are clear. The progress of the software can be seen quite often and in case of bugs in the last release than the previous version can be restored at any time. This gives safety in terms of saving the current status, and then adjust or develop it in accordance with the requirements that are dynamic in most cases. Consequently, if there should be any changes in the final release of the software, then they can be easily applied.

In HappyFace, we practice agile software development principles. As HappyFace has a modular structure, different modules can be modified or designed independently. After a certain period of time, all developed modules are merged together and the current release is ready.

Appendix B

Installing HappyFace and Building an Instance

B.1 Required Packages

All required packages for installation of HappyFaceCore and ATLAS modules are listed below:

- *python >= 2.6*
- *httpd >= 2.0*
- *python-cherrypy >= 3.0*
- *python-sqlalchemy >= 0.5*
- *python-migrate*
- *python-mako*
- *python-matplotlib*
- *python-beautifulsoup*
- *python-sqlite*
- *python-psycopg2*
- *python-lxml*
- *numpy*
- *mod_wsgi*
- *sqlite*

In addition to above mentioned packages, the grid-enabled HappyFace extension has its own requirements which are listed below:

- *openssl*
- *cvmfs*

- *eclipse-platform*
- *eclipse-pde*
- *yum-protectbase*
- *fetch-crl*
- *ca-policy-egi-core*
- *umd-release >= 3.0.0*
- *rpmforge-release*
- *kdesdk*

HappyFace uses a Linux distribution, in particular, Scientific Linux CERN 6 (SLC6).

B.2 Setting up the CVMFS Environment

The CernVM File System [134] or mostly known as the CVMFS is a file system that provides a reliable, scalable and efficient software distribution service.

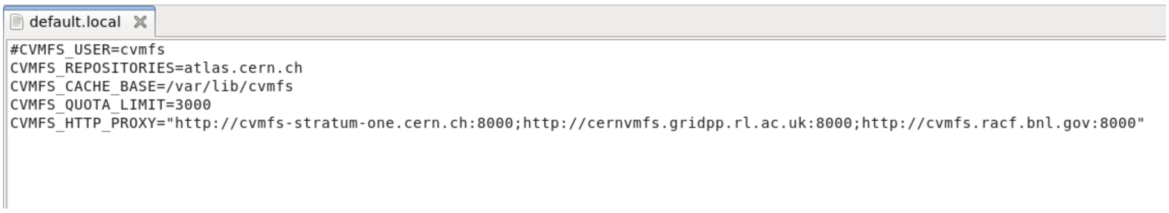
It is providing a complete and portable environment for running and developing LHC data analysis in the grid and in the laptop/desktop computer. It is working independently of the operating system (Linux, Windows, MacOS). The CVMFS is similar to POSIX read-only file system on user space. Files and directories are hosted on standard web servers and mounted in the universal namespace */cvmfs*.

The CVMFS is widely used for HEP experiments. It allows physicists working on LHC experiments at CERN to use their computers effectively.

In many cases, the CVMFS replaces package managers and repositories on cluster file systems. For the LHC experiments, the CVMFS hosts several millions of files and directories that are distributed to thousands of client computers.

In ATLAS, in order to use ATLAS environments, the CVMFS is required.

The installation or setting up the CVMFS is very simple. There are two files *default.local* and *install_cvmfs.sh* Linux shell script. In the *default.local* file (Figure B.1) is mentioned all necessary paths and the *install_cvmfs.sh* Linux shell script (Figure B.2) is just installing the CVMFS environment. These two files should be located in the same folder.



```
#CVMFS_USER=cvmfs
CVMFS_REPOSITORIES=atlas.cern.ch
CVMFS_CACHE_BASE=/var/lib/cvmfs
CVMFS_QUOTA_LIMIT=3000
CVMFS_HTTP_PROXY="http://cvmfs-stratum-one.cern.ch:8000;http://cernvmfs.gridpp.rl.ac.uk:8000;http://cvmfs.racf.bnl.gov:8000"
```

Figure B.1: *default.local* file.

If the CVMFS is succesfully installed the *ATLAS user environment* will be available. In order to check it, the following commands are needed to be executed from the Linux terminal.

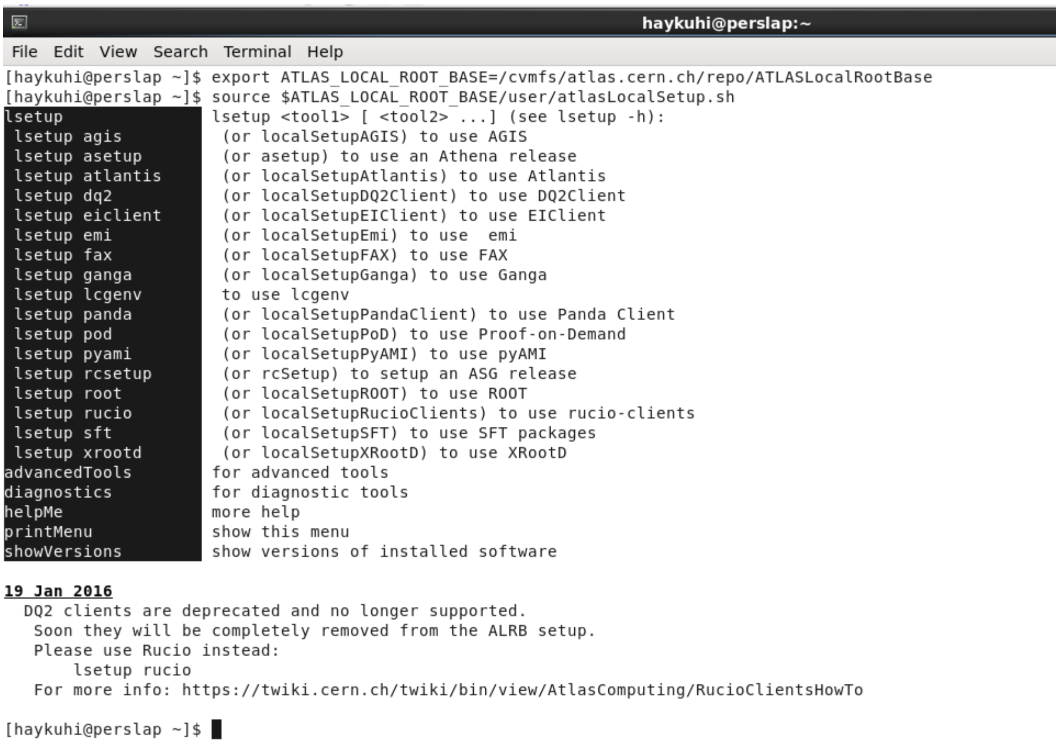
```
#!/bin/sh

#----------------------------------
# Check CVMFS
#----------------------------------
rpm -q cvmfs && exit 0



#----------------------------------
# make yum repo
#----------------------------------
repobase="http://cvmrepo.web.cern.ch/cvmrepo/yum/cvmfs/EL/6/x86_64"
repofile="cvmfs-release-2-4.el6.noarch.rpm"
HOME=/root

if ! [ -e $HOME/$repofile ]; then
    wget $repobase/$repofile -O $HOME/$repofile
    yum -y localinstall $HOME/$repofile

    #----------------------------------
    # install cvmfs
    #----------------------------------
    yum clean all
    yum -y install cvmfs cvmfs-auto-setup cvmfs-init-scripts
fi
#----------------------------------
# Configuration
#----------------------------------
cp -v default.local /etc/cvmfs/default.local
#----------------------------------
# Start
#----------------------------------
/etc/init.d/autofs restart

#----------------------------------
# Check
#----------------------------------
cvmfs_config chksetup
cvmfs_config probe

# details
cvmfs_config stat -v
cvmfs_talk cache list

# ls
ls /cvmfs/atlas.cern.ch
```

Figure B.2: *install_cvmfs.sh* Linux shell script.

```
$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
```

```
$ source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
```

After successful installation, all ATLAS environments will be listed, as it is shown in Figure B.3.



```
[haykuhi@perslap ~]$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
[haykuhi@perslap ~]$ source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
lsetup               lsetup <tool1> [ <tool2> ...] (see lsetup -h):
 lsetup agis          (or localSetupAGIS) to use AGIS
 lsetup asetup        (or asetup) to use an Athena release
 lsetup atlantis      (or localSetupAtlantis) to use Atlantis
 lsetup dq2           (or localSetupDQ2Client) to use DQ2Client
 lsetup eiclient      (or localSetupEIClient) to use EIClient
 lsetup emi           (or localSetupEmi) to use  emi
 lsetup fax           (or localSetupFAX) to use FAX
 lsetup ganga         (or localSetupGanga) to use Ganga
 lsetup lcgenv        to use lcgenv
 lsetup panda         (or localSetupPandaClient) to use Panda Client
 lsetup pod           (or localSetupPoD) to use Proof-on-Demand
 lsetup pyami         (or localSetupPyAMI) to use pyAMI
 lsetup rcsetup       (or rcSetup) to setup an ASG release
 lsetup root          (or localSetupROOT) to use ROOT
 lsetup rucio         (or localSetupRucioClients) to use rucio-clients
 lsetup sft           (or localSetupSFT) to use SFT packages
 lsetup xrootd        (or localSetupXRootD) to use XRootD
advancedTools        for advanced tools
diagnostics          for diagnostic tools
helpMe               more help
printMenu            show this menu
showVersions         show versions of installed software

19 Jan 2016
  DQ2 clients are deprecated and no longer supported.
   Soon they will be completely removed from the ALRB setup.
   Please use Rucio instead:
       lsetup rucio
   For more info: https://twiki.cern.ch/twiki/bin/view/AtlasComputing/RucioClientsHowTo

[haykuhi@perslap ~]$ █
```

Figure B.3: Available ATLAS environments, after executing above mentioned commands.

B.3 Installation

The source code of HappyFace is stored in the *GitHub* repository and is located at the *https://github.com/HappyFaceGoettingen/* url.

The *GitHub* [141] is a web-based *Git* [141] repository. The *Git* is a source code management system used for software development. While the *Git* repository is a command-line tool, the *GitHub* has a graphical interface and has the same functionality as the *Git* repository.

The HappyFace *GitHub* repository consists of several repositories, listed below:

- *Red-Comet*: stores the source code of the grid-enabled HappyFace extension.
- *HappyFaceATLASModules*: stores the source code of the HappyFace ATLAS modules.

- *HappyFaceExtra*: stores the source code of the HappyFace non-ATLAS modules, such as CREAMCE, Ganglia, PBS modules.
- *HappyFaceWebService*: stores the source code of the HappyFace web services such as WSDL and RESTFull web services.
- *HappyFaceSmartPhoneApp*: stores the source code of the HappyFace smartphone application
- *HappyFaceCore*: stores the source code of the HappyFace core modules.
- *HappyFace-Integration-Server*: stores script files through which it is possible to create RPM packages [142] of all mentioned above repositories. It contains specification files, source files and all other necessary files for creating RPM packages.

Before installing the HappyFace software the *RPMforge* needs to be installed.

RPMforge [143] is a repository that provides more than 5000 software packages in the rpm format for Red Hat Enterprise Linux (RHEL) and Community ENTerprise Operating System (CentOS) Linux distributions.

```
## RHEL/CentOS 6 64 Bit OS ##
```

```
$ wget http://packages.sw.be/rpmforge-release/rpmforge-release-0.5.2-2.el6.rf.x86_64.rpm
```

```
$ rpm -Uvh rpmforge-release-0.5.2-2.el6.rf.x86_64.rpm
```

After a successful installation of the *rpmforge-release* package the HappyFace software can be installed.

Installation of HappyFace is simple by using RPM packages.

```
$ yum install HappyFaceCore-3.0.0-2.x86_64.rpm
```

- the command installs HappyFace Core

```
$ yum install HappyFace-ATLAS-3.0.0-20160318.x86_64.rpm
```

- the command installs the HappyFace ATLAS modules

```
$ yum install HappyFace-Red-Comet-3.0.0-20150611.x86_64.rpm
```

- the command installs the HappyFace grid-enabled extension.

After having all necessary RPM packages successfully installed, the X.509 user certificate and key needs to be generated or copied to the */var/lib/HappyFace/cert*.

Steps of generating the X.509 user certificate and key.

1. Generate user certificate

```
$ openssl pkcs12 -clcerts -nokeys -in usercert.p12 -out usercert.pem
```

2. Create a private certficate with passphrase

```
$ openssl pkcs12 -nocerts -in usercert.p12 -out userkey.pem
```

3. Create a private certificate without passphrase

```
$ openssl rsa -in userkey.pem -out userkey.nopass.pem
```

4. Change permissions

```
$ chmod 400 userkey.nopass.pem
$ chmod 644 usercert.pem
```

5. Copy a grid user certificate

```
$ cp -v userkey.nopass.pem /var/lib/HappyFace/cert
$ cp -v usercert.pem /var/lib/HappyFace/cert
```

B.4 Running an Instance

After a successful installation of the HappyFace instance on the machine, the HappyFace folder will be available at the */var/lib/HappyFace* path. To run an instance is very simple, only one file, the *acquire.py* needs to be executed and after completion of the output will be displayed on the web interface.

```
$ python acquire.py
```

If there is a need to update the database structure, the following command needs to be executed.

```
$ python tools.py dbupdate
```

HappyFace is a generic tool and easy to configure. For having HappyFace working in any computing site, modifications in the HappyFace configuration file (*happyface.cfg*) B.5.1 and module configuration files are needed.

For grid-enabled HappyFace modules (GridFTP, GridSRM and ATLAS DDM) configuration files (*grid_ftp_module.cfg*, *grid_srm_module.cfg* and *atlas_ddm_module.cfg*) are displayed in Figures B.5.2, B.5.3 and B.5.4 respectively.

B.5 Files

B.5.1 happyface.cfg

```
[paths]
happyface_url = /

static_dir = static
archive_dir = %(static_dir)s/archive

tmp_dir = %(static_dir)s/tmp

hf_template_dir = templates
module_template_dir = modules/templates
template_cache_dir = mako_cache

template_icons_url = %(static_url)s/themes/armin_box_arrows

local_happyface_cfg_dir = config
category_cfg_dir = config/categories-enabled
module_cfg_dir = config/modules-enabled

acquire_logging_cfg = defaultconfig/acquire.log
render_logging_cfg = defaultconfig/render.log

# NOTE: Changing these URLs might have limited effect
static_url = %(happyface_url)sstatic
archive_url = %(static_url)s/archive


[happyface]
# colon separated list of categories, if empty
# all are used in a random order. The name here
# corresponds to the section name.
categories =
stale_data_threshold_minutes = 60

# automatic reload interval in minutes
reload_interval = 15

[auth]
# A file containing authorized DNs to access the site.
# One DN per line
dn_file =

# If the given DN is not found in the file above, if any, the following
# script is called with DN as first argument.
# The script must return 1 if user has access, 0 otherwise.
auth_script =

[template]
# relative to static URL
logo_img = /images/default_logo.jpg
documentation_url = https://ekptrac.physik.uni-karlsruhe.de/trac/HappyFace
web_title = HappyFace Project

[database]
url = sqlite:///HappyFace.db

[downloadService]
timeout = 300
global_options =
[plotgenerator]
enabled = False
backend = agg


[global]

server.socket_host: "0.0.0.0"
server.socket_port: 8080

tools.encode.on: True
tools.encode.encoding: "utf-8"
tools.decode.on: True
tools.trailing_slash.on: True


[grid]
enabled = True
# False = grid-proxy-init, True = voms-proxy-init
voms.enabled = True
vo = atlas
proxy.renew.enabled = True
proxy.lifetime.threshold.hours = 1
proxy.valid.hours = 100
# with voms-proxy-init
proxy.voms.hours= 100
x509.cert.dir = /etc/grid-security/certificates
#x509.cert.dir =
/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase/etc/grid-security-emi/certificates
x509.user.cert = /var/lib/HappyFace/cert/usercert.pem
x509.user.key = /var/lib/HappyFace/cert/userkey.pem
x509.user.proxy = /var/lib/HappyFace/cert/x509up_happyface


[cvmfs]
enabled = True
rucio.account = hmushegh
agis = False
atlantis = False
rucio.client = False
emi = True
fax = False
ganga = False
gcc = False
pacman = False
panda.client = False
pyami = False
pod = False
root = False
dq2wrappers = False
sft = False
xrootd = False
```

B.5.2 grid_ftp_module.cfg

```
[grid_ftp]
module = GridFtpCopyViewer
name = Grid FTP Viewer
type = rated
weight = 1.0


timeout = 240
plot_start_date = 2015-06-05

site1_host = se-goegrid.gwdg.de
site1_name = GoeGrid-GoeGrid
site1_port = 8443
site1_space_token_scratchdisk = GOEGRID_SCRATCHDISK
site1_space_token_localgroupdisk = GOEGRID_LOCALGROUPDISK
site1_space_token_proddisk = GOEGRID_PRODDISK
site1_space_token_datadisk = GOEGRID_DATADISK
site1_scratchdisk_path = /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test_haykuhi/
site1_localgroupdisk_path =
/pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test_haykuhi/
site1_proddisk_path = /pnfs/gwdg.de/data/atlas/atlasproddisk/
site1_datadisk_path = /pnfs/gwdg.de/data/atlas/atlasdatadisk/



site2_host = grid-se.physik.uni-wuppertal.de
site2_name1 = GoeGrid-Wuppertal
site2_port = 8443
site2_name2 = Wuppertal-GoeGrid
site2_space_token_scratchdisk = WUPPERTAL_SCRATCHDISK
site2_space_token_localgroupdisk = WUPPERTAL_LOCALGROUPDISK
site2_space_token_proddisk = WUPPERTAL_PRODDISK
site2_space_token_datadisk = WUPPERTAL_DATADISK
site2_scratchdisk_path =
/pnfs/physik.uni-wuppertal.de/data/atlas/atlasscratchdisk/test_haykuhi/
site2_localgroupdisk_path =
/pnfs/physik.uni-wuppertal.de/data/atlas/atlaslocalgroupdisk/test_haykuhi/
site2_proddisk_path = /pnfs/physik.uni-wuppertal.de/data/atlas/atlasproddisk/
site2_datadisk_path = /pnfs/physik.uni-wuppertal.de/data/atlas/atlasdatadisk/


site3_host = atlas.bu.edu
site3_name1 = GoeGrid-BU_ATLAS_Tier2
site3_port = 8443
site3_name2 = BU_ATLAS_Tier2-GoeGrid
site3_space_token_scratchdisk = BU_ATLAS_Tier2_SCRATCHDISK
site3_space_token_localgroupdisk = BU_ATLAS_Tier2_LOCALGROUPDISK
site3_space_token_proddisk = BU_ATLAS_Tier2_PRODDISK
site3_space_token_datadisk = BU_ATLAS_Tier2_DATADISK
site3_scratchdisk_path = /gpfs1/atlasscratchdisk/test_haykuhi/
site3_localgroupdisk_path = /gpfs1/atlaslocalgroupdisk/test_haykuhi/
site3_proddisk_path = /gpfs1/atlasproddisk/
site3_datadisk_path = /gpfs1/atlasdatadisk/
```

B.5.3 grid_srm_module.cfg

```
[grid_srm]
module = GridSRMCopyViewer
name = Grid SRM Copy Viewer
type = rated
weight = 1.0


timeout = 240
plot_start_date = 2015-06-10

site1_host = se-goegrid.gwdg.de
site1_name = GoeGrid-GoeGrid
site1_port = 8443
site1_space_token_scratchdisk = GOEGRID_SCRATCHDISK
site1_space_token_localgroupdisk = GOEGRID_LOCALGROUPDISK
site1_space_token_proddisk = GOEGRID_PRODDISK
site1_space_token_datadisk = GOEGRID_DATADISK
site1_scratchdisk_path = /pnfs/gwdg.de/data/atlas/atlasscratchdisk/test_haykuhi/
site1_localgroupdisk_path =
/pnfs/gwdg.de/data/atlas/atlaslocalgroupdisk/test_haykuhi/
site1_proddisk_path = /pnfs/gwdg.de/data/atlas/atlasproddisk/
site1_datadisk_path = /pnfs/gwdg.de/data/atlas/atlasdatadisk/



site2_host = grid-se.physik.uni-wuppertal.de
site2_name1 = GoeGrid-Wuppertal
site2_port = 8443
site2_name2 = Wuppertal-GoeGrid
site2_space_token_scratchdisk = WUPPERTAL_SCRATCHDISK
site2_space_token_localgroupdisk = WUPPERTAL_LOCALGROUPDISK
site2_space_token_proddisk = WUPPERTAL_PRODDISK
site2_space_token_datadisk = WUPPERTAL_DATADISK
site2_scratchdisk_path =
/pnfs/physik.uni-wuppertal.de/data/atlas/atlasscratchdisk/test_haykuhi/
site2_localgroupdisk_path =
/pnfs/physik.uni-wuppertal.de/data/atlas/atlaslocalgroupdisk/test_haykuhi/
site2_proddisk_path = /pnfs/physik.uni-wuppertal.de/data/atlas/atlasproddisk/
site2_datadisk_path = /pnfs/physik.uni-wuppertal.de/data/atlas/atlasdatadisk/


site3_host = atlas.bu.edu
site3_name1 = GoeGrid-BU_ATLAS_Tier2
site3_port = 8443
site3_name2 = BU_ATLAS_Tier2-GoeGrid
site3_space_token_scratchdisk = BU_ATLAS_Tier2_SCRATCHDISK
site3_space_token_localgroupdisk = BU_ATLAS_Tier2_LOCALGROUPDISK
site3_space_token_proddisk = BU_ATLAS_Tier2_PRODDISK
site3_space_token_datadisk = BU_ATLAS_Tier2_DATADISK
site3_scratchdisk_path = /gpfs1/atlasscratchdisk/test_haykuhi/
site3_localgroupdisk_path = /gpfs1/atlaslocalgroupdisk/test_haykuhi/
site3_proddisk_path = /gpfs1/atlasproddisk/
site3_datadisk_path = /gpfs1/atlasdatadisk/
```

B.5.4 atlas_ddm_module.cfg

```
[ddm_datasets_viewer]
module = DdmDatasetsViewer
name = Atlas DDM Datasets Viewer
description =
instruction =
type = rated
weight = 1.0


# Parameters
datasets_range_start = 1
datasets_range_end = 50
dataset_size = 50

space_token = GOEGRID_LOCALGROUPDISK
limit_table_size = 50
```

Bibliography

[1] C. J. Rhodes, *Large Hadron Collider (LHC)*, Sc. Prog. **96.1** (2013): 95-109.

[2] `LHC RUN 2.`

[3] P. Higgs, *Broken symmetries, massless particles and gauge fields*, Phys. Lett. **12** (1964) 132–133.

[4] S. Dimopoulos and H. Georgi, *Softly broken supersymmetry and SU (5)*, Nucl. Phys. B **193** (1981) 150–162.

[5] S. L. Glashow, *Partial-symmetries of weak interaction*, Nucl. Phys. **22** (1961) 579–588.

[6] J. Goldstone, A. Salam and S. Weinberg, *Broken Symmetries*, Phys. Rev. **127** (1962) 965-970.

[7] A. Salam, et al., *Electromagnetic and weak interactions*, Phys. Lett. **13** (1964) 168–171.

[8] T. Kinoshita, *Quantum electrodynamics*, Vol. **7** World Scientific (1990).

[9] W. Marciano and H. Pagels, *Quantum chromodynamics*, Phys. Rep. **36.3** (1978): 137-276.

[10] V. Flaminio and B. Saitta, *Neutrino oscillation experiments*, La Rivista del Nuovo Cimento (1978-1999) **10.8** (1987): 1-126.

[11] FM. Liu and SX. Liu, *Quark-gluon plasma formation time and direct photons from heavy ion collisions*, Phys. Rev. C **89.3** (2014): 034906.

[12] L. Evans and P. Bryant, *LHC machine*, J. of Inst. **3.08** (2008): S08001.

[13] J. Shiers, *The worldwide LHC computing grid (worldwide LCG)*, Comput. Phys. Commun. **177.1** (2007): 219-223.

[14] R.Cashmore, et al., *Prestigious Discoveries at CERN: 1973 Neutral Currents. 1983 W Z Bosons*, Spr. Sc. Business Media, (2004).

[15] The ATLAS Collaboration, CDF Collaboration, CMS Collaboration, D0 Collaboration, *First combination of Tevatron and LHC measurements of the top-quark mass*, `arXiv:1403.4427`.

[16] K. A. Olive and Particle Data Group, *Review of particle physics*, Chinese Phys. C **38.9** (2014): 090001.

[17] H. Georgi and S. L. Glashow, *Unity of all elementary-particle forces*, Phys. Rev. Lett. **32.8** (1974): 438.

[18] F. Englert and R. Brout, *Broken Symmetry and the Mass of Gauge Vector Mesons*, Phys. Rev. Lett. **13** (1964) 321–323.

[19] K. A. Olive, et al., *Review of Particle Physics, 2014-2015*, Chin. Phys. C **38** (2014) 090001, `arXiv:1753419`.

[20] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, J. of Inst. Vol. **3** (2008).

[21] Courtesy of the ATLAS experiment.

[22] ATLAS collaboration, *Technical Design Report for the ATLAS Forward Proton Detector*, Rep. Nr.: CERN-LHCC-2015-009 (2015) 81-85.

[23] M. Borodin, et al., *Unified System for Processing Real and Simulated Data in the ATLAS Experiment* (2015), `arXiv:1508.07174`.

[24] I. Foster and C. Kesselman, eds., *The Grid 2: Blueprint for a new computing infrastructure*, Els. Ser. (2003).

[25] I. G. Bird, *LHC computing (WLCG): Past, present, and future*, Grid and Cloud Comp. Conc. and Pract. App. **192** (2016): 1.

[26] R. W. L. Jones and D. Barberis, *The evolution of the ATLAS computing model*, J. of Phys.: Conf. Ser. Vol. **219** No. 7. IOP Publ. (2010).

[27] G. Ambrosini, et al., *The ATLAS DAQ and Event Filter prototype " 1" project*, Comp. Phys. Commun. **110.1** (1998): 95-102.

[28] S. Murray, et al., *Tape write-efficiency improvements in CASTOR*, J. of Phys.: Conf. Ser. **396** (2012) 042042, `arXiv:1485674`.

[29] F. Berman, et al., *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, (2003) 1 edition ISBN: 978-0-470-85319-1.

[30] I. Foster, et al., 2001, *The anatomy of the grid: Enabling scalable virtual organizations* Intern. j. of high perf. comp. app. **15.3** (2001): 200-222.

[31] M. A. Murphy, et al., *Virtual Organization Clusters*, (2009) 17th Eu. Int. Conf. on Paral., Distr. and Net.-based Process., IEEE pp. 401-408.

[32] I. Foster, et al., *What is the grid?-a three point checklist*, GRIDtoday, Vol. 1, No. **6** (2002).

[33] I. Foster, *Globus toolkit version 4: Software for service-oriented systems*, IFIP Int. Conf. on Net. and Par. Comp. Springer. Berlin Heidelberg, (2005).

[34] B. Ross, et al., *Stronger Password Authentication Using Browser Extensions*, Proc. of the 14th USENIX Secur. Symp. (2005).

[35] S. T. F. Al-Janabi and M. A. Rasheed, *Public-key cryptography enabled Kerberos authentication*, Developments in E-systems Engineering (DeSE) IEEE (2011).

[36] R. Housley and T. Polk *Planning for PKI: best practices guide for deploying public key infrastructure*, John Wiley Sons, Inc., (2001).

[37] S. Santesson, et al., *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, No. RFC **6960**, (2013).

[38] A. Kahate, *Cryptography and network security*, Tata McGraw-Hill Edu. (2013).

[39] R. L. Rivest, et al., *Cryptographic communications system and method*, U.S. Pat. No. 4,405,829 (1983).

[40] K. R. Vogel, et al., *Updating trusted root certificates on a client computer*, U.S. Pat. No. 6,816,900 (2004).

[41] E. Rescorla, *SSL and TLS: designing and building secure systems*, Vol. **1** Reading: Addison-Wesley (2001).

[42] W. Stallings, *Cryptography and network security: principles and practices*, Pearson Edu. India, Section **14.2** (2006).

[43] V. Welch, et al. *X. 509 proxy certificates for dynamic delegation*, 3rd ann. PKI RD workshop. Vol. **14** (2004).

[44] S. Tuecke, et al., *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*, No. RFC **3820** 2004.

[45] J. Gilbert and R. Perry, *A Java API for X. 509 Proxy Certificates*, HP Laboratories, HPL-2008-77 (2008).

[46] L. Pearlman, et al., *The Community Authorization Service: Status and Future*, CHEP03, La Jolla, California (2003).

[47] R. Alfieri, et al., *VOMS, an Authorization System for Virtual Organizations*, Grid comp. Springer Berlin Heidelberg, (2004).

[48] B. Plale, et al., *Key concepts and services of a grid information service*, Proc. of the 15th Int. Conf. on Paral. and Distr. Comp. Sys. (2002).

[49] J. S. Bowman, et al., *The practical SQL handbook: using structured query language*, Addison-Wesley Longman Publ. Co. Inc. (1996).

[50] T. Howes and M. Smith. *LDAP: programming directory-enabled applications with lightweight directory access protocol*, Que Publ. **1** ed. (1997) ISBN-10: 1578700000.

[51] V. Hernández, *Challenges and opportunities of healthgrids*, Proc. of HealthGrid Vol. **120**. IOS Press (2006).

[52] A. Cooke, et al., *R-GMA: An information integration system for grid monitoring*, OTM Confed. Int. Conf. Springer Berlin Heidelberg (2003).

[53] S. Zhang, et al., *Dynamic VM Provisioning for TORQUE in a Cloud Environment*, J. of Phys.: Conf. Ser. Vol. **513** No. 3. IOP Publ. (2014).

[54] D. Thain, et al, *Condor and the Grid*, Grid Comp.: Making the Global Infrast. A Reality (2003): **299-335**.

[55] G. Ma and P. Lu, *PBSWeb: A Web-based Interface to the Portable Batch System*, 12th IASTED Int. Conf. on Paral. and Distr. Comp. and Sys. (PDCS), Las Vegas, Nevada, USA (2000).

[56] G. Staples, *TORQUE resource manager*, Proc. of the 2006 ACM/IEEE Conf. on Supercomp. (2006).

[57] I. Lumb and C. Smith, *Scheduling Attributes and Platform LSF*, Grid resource management. Springer US, (2004) **171-182**.

[58] K. Subramanian, et al., *Workload Management with LoadLeveler*, IBM Redbooks (2001) ISBN: 978-0738422091.

[59] B. Stockebrand, *Quality of Service (QoS)*, IPv6 in Pract.: A Unixer's Guide to the Next Gener. Internet (2007): **327-331**.

[60] A. Shoshani, et al., *Storage resource managers*, Grid resource management. Springer US, **321-340** (2004).

[61] A. Sim and A. Shoshani, *The Storage Resource Manager Interface Specification Version 2.2*, (2009) `SRMv2.2`.

[62] M. Cannataro, et al., *Evaluating and enhancing the use of the GridFTP protocol for efficient data transfer on the grid*, Rec. Adv. in Paral. Virt. Mach. and Msg. Passing Interface, Springer Berlin Heidelberg (2003).

[63] `globus-toolkit-command`.

[64] `UberFTP`

[65] W. Allcock, et al., *GridFTP: Protocol extensions to FTP for the Grid*, Global Grid ForumGFD-RP (2003) **20**: 1-21, `GFD.20` .

[66] B. Allcock, et al., *GridFTP v2 protocol description*, GridFTP Working Group, Tech. Rep. (2005), `GFD.47` .

[67] F. Magoules, et al., *Introduction to Grid Computing*, By CRC Press (2009) ISBN: 9781420074062.

[68] D. Talia, et al., *Grid Middleware and Services*, Springer Singapore Pte. Limited (2009).

[69] S. Burke, et al., *GLITE 3.2 USER GUIDE*, Document identifier: CERN-LCG-GDEIS-722398 (2012), `gLite-3-UserGuide`.

[70] M. Romberg, *The UNICORE grid infrastructure*, Scientific Programming **10.2**: 149-157 (2002).

[71] E. Laure, *Programming the Grid with gLite*, Comp. methods in sc. and tech. **12.1** (2006): 33-45.

[72] L. Wang, et al., *Grid Computing: Infrastructure, Service, and Applications*, CRC Press (2009) ISBN: 9781420067668.

[73] R. Ramakrishnan, et al., *Database management systems*, Osborne/McGraw-Hill (2000).

[74] L.V. Shamardin, *LCG/gLite BDII performance measurements*, PoS (2007): **014** .

[75] P. Andreetto, et al., *CREAM: a simple, grid-accessible, job management system for local computational resources* CHEP 2006, Mumbay, India (2006).

[76] C. Wang, et al., *Hybrid checkpointing for MPI jobs in HPC environments*, Paral. and Distr. Sys. (ICPADS), IEEE 16th Int. Conf. (2010).

[77] P. Fuhrmann and V. Gülzow, *dCache, storage system for the future*, EU Conf. on Para. Process. Springer Berlin Heidelberg, (2006).

[78] J. Meyer, et al., *ATLAS Tier-2 at the Compute Resource Center GoeGrid in Göttingen*, J. of Phys.: Conf. Ser. Vol. 331. No. **7** IOP Publ (2011).

[79] D. Jackson, et al., *Core algorithms of the Maui scheduler*, Workshop on Job Sched. Strat. for Paral. Process. Springer Berlin Heidelberg (2001).

[80] GWDG SAN storage system `GWDG_SAN_system`.

[81] P. Rey, et al., *The Accounting Infrastructure in EGEE*, Proc. of the 1st Ib. Grid Infrast. Conf., Santiago de Compostela, Spain (2007).

[82] Rutherford Appleton Laboratory (RAL), `RAL`.

[83] W. Barth, *Nagios: System and network monitoring*, No Starch Press Series (2008), ISBN: 1593271794, 9781593271794.

[84] P.M. Papadopoulos, et al., *NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters*, Concur. and Comput.: Pract. and Exp. **15.7-8** (2003): 707-725.

[85] M. P. DeHaan, *Software provisioning in multiple network configuration environment* , U.S. Patent No. 9,021,470. (2015).

[86] M. L. Massie, et al., *The ganglia distributed monitoring system: design, implementation, and experience*, Paral. Comput. **30.7** (2004): 817-840.

[87] M. Burgess, *Recent developments in cfengine*, The Hague (2001).

[88] `Scientific Linux CERN 6 (SLC6).`

[89] R. W. L. Jones and D. Barberis, *The evolution of the ATLAS computing model*, J. of Phys.: Conf. Ser. Vol. 219. No. **7** IOP Publ. (2010).

[90] A. Anisenkov, et al., *AGIS: the ATLAS grid information system*, J. of Phys.: Conf. Ser. Vol. 513. No. **3** IOP Publ. (2014).

[91] J. Schovancová, *ATLAS Distributed Computing* , No. ATL-SOFT-SLIDE-2011-551. ATL-COM-SOFT-2011-022 (2011).

[92] G. Mathieu, et al., *GOCDB, a topology repository for a worldwide grid infrastructure*, J. of Phys.: Conf. Ser. Vol. 219. No. **6** IOP Publ. (2010).

[93] R. Quick, et al., *RSV: OSG Fabric Monitoring and Interoperation with WLCG Monitoring Systems*, Manuscript in Proc. of 17th Int. Conf. on Comp. in HEP (2009).

[94] J. Forcier, et al., *Python web development with Django*, Addison-Wesley Prof. (2008).

[95] J. P. Baud, et al., *LCG Data Management: From EDG to EGEE*, UK eSc. All Hands Meeting Proc., Nottingham, UK (2005).

[96] R. K. Madduri, et al., *Reliable file transfer in grid environments*, Local Computer Networks, Proceedings. LCN 27th Ann. IEEE Conf. (2002).

[97] T. Perelmutov, et al., *Storage Resource Manager*, Proc. of CHEP'04 (2004).

[98] D. Dykstra and L. Lueking, *Greatly improved cache update times for conditions data with Frontier/Squid*, J. of Phys.: Conf. Ser. Vol. 219. No. **7** IOP Publ. (2010).

[99] D. Dykstra and J. Blomer, *Security in the CernVM File System and the Frontier Distributed Database Caching System*, J. of Phys.: Conf. Ser. Vol. 513. No. **4** IOP Publ. (2014).

[100] B. Tierney, et al., *perfsonar: Instantiating a global network measurement framework*, SOSP Wksp. Real Overlays and Distrib. Sys (2009).

[101] `AGIS web interface for runing services`.

[102] `AGIS web interface for pandaqueues`.

[103] T. Maeno, et al., *Overview of atlas panda workload management*, J. of Phys.: Conf. Ser. Vol. 331. No. **7** IOP Publ. (2011).

[104] P. Nilsson, et al., *The ATLAS PanDA Pilot in Operation*, J. of Phys.: Conf. Ser. Vol. 331. No. **6** IOP Publ. (2011).

[105] PanDA web interface for GoeGrid `Panda`$_{GoeGrid}$.

[106] V. Garonne, et al., *Rucio–The next generation of large scale distributed system for ATLAS Data Management*, J. of Phys.: Conf. Ser. Vol. 513. No. **4** IOP Publ. (2014).

[107] `Rucio account`.

[108] `Rucio architecture`.

[109] S. Albrand, et al., *The ATLAS metadata interface*, J. of Phys.: Conf. Ser. Vol. 219. No. **4** IOP Publ. (2010).

[110] `DDM Dashboard`.

[111] `DDM Dashboard Transfer Plots`.

[112] De, Kaushik, et al., *ATLAS Distributed Computing Operations Shift Team Experience*, J. of Phys. Conf. Ser. Vol. 331. No. **7** (2011).

[113] J. Andreeva, et al., *Automating ATLAS Computing Operations using the Site Status Board*, J. of Phys.: Conf. Ser. Vol. 396. No. **3** IOP Publ. (2012).

[114] S. Campana, et al., *Evolving ATLAS computing for today's networks*, J. of Phys.: Conf. Ser. Vol. 396. No. **3** IOP Publ. (2012).

[115] `Site Status Board web interface (shifter view)`.

[116] ATLAS Collaboration, et al., *ATLAS high-level trigger, data acquisition and controls technical design report*, ATLAS Technical Design Reports (2003).

[117] J. Zhang, *ATLAS data acquisition*, Real Time Conf. 16th IEEE-NPSS. IEEE (2009).

[118] T. Cornelissen, et al., *Concepts, design and implementation of the ATLAS New Tracking (NEWT)*, No. ATL-COM-SOFT-2007-002. (2007).

[119] I. Antcheva, et al., *ROOT—A C++ framework for petabyte data storage, statistical analysis and visualization*, Comp. Phys. Commun. 180.12 (2009): 2499-2512.

[120] J. S. Wholey, *Monitoring performance in the public sector: Future directions from international experience*, Eds. J. Mayne and E. Zapico-Goñi, Transaction Publ. (2007).

[121] S. Ligus, *Effective Monitoring and Alerting*, O'Reilly Media Print (2002) ISBN: 978-1-4493-3352-2.

[122] H. Musheghyan, et al., 2015, *HappyFace as a generic monitoring tool for HEP experiments*, J. of Phys.: Conf. Ser. Vol. 664. No. **6** IOP Publ. (2015).

[123] X. Zhang and J. M. Schopf, *Performance analysis of the globus toolkit monitoring and discovery service, mds2*, Perf. Comput. and Commun. (2004) Int. Conf. on. IEEE.

[124] V. Mauch, et al., *The HappyFace project*, J. of Phys.: Conf. Ser. Vol. 331. No. **8** IOP Publ. (2011).

[125] J. Highsmith and A. Cockburn *Agile software development: The business of innovation*, Computer **34.9** (2001): 120-127.

[126] `Python Mako Templates`.

[127] M. Owens and G. Allen, *SQLite*, Apress LP (2010).

[128] `Python Matplotlib`.

[129] `Linux cron job`.

[130] L. O. Gerlach, *Update of HappyFace modules for Atlas*, BSc thesis Nr.: II.Physik-UniGö-BSc-2015/07 (2015).

[131] E. Buschmann, *Module developments on the HappyFace project for the ATLAS Grid Computing*, BSc thesis Nr.: II.Physik-UniGö-BSc-2014/02 (2014).

[132] C. G. Wehrberger, *HappyFace Meta-Monitoring for ATLAS in the Worldwide LHC Computing Grid*, MSc thesis Nr.: II.Physik-UniGö-MSc-2013/07 (2013).

[133] J. Blomer, et al., *Status and future perspectives of CernVM-FS*, J. of Phys.: Conf. Ser. Vol. 396. No. **5** IOP Publ. (2012).

[134] J. Blomer, et al., *Distributing LHC application software and conditions databases using the CernVM file system*, J. of Phys.: Conf. Ser. Vol. 331. No. **4** IOP Publ. (2011).

[135] M. Blaha and J. Rumbaugh, *Object-oriented modeling and design with UML*, Prentice Hall (2005).

[136] R. Verma and C. Giridhar, *Design Patterns in Python*, Publ. by Testing Persp. (2011).

[137] J. Rumbaugh, et al., *Unified Modeling Language Reference Manual*, Pearson Higher Edu. (2004) ISBN: 0321245628.

[138] `Umbrello modelling tool`.

[139] `UberFTP CLI`.

[140] `SRM CLI`.

[141] S. Chacon and B. Straub, *Pro git*, Apress (2014), `Pro Git`.

[142] `How to create an RPM package`.

[143] R. Petersen, *Red Hat Enterprise Linux 6: Desktop and Administration*, Surfing Turtle Press (2011).

Acknowledgements

I would like to express my deep thankfulness and respect to Prof. Dr. Arnulf Quadt for giving me this unique opportunity to come to Germany and to start my PhD at II Institute of Physics in University of Goettingen. I am very much thankful to him for his guidance and support during the whole time period of my PhD. Also for all chances to participate in different workshops, conferences, meetings, summer schools, through which I have managed to meet and get to know many other people and to improve my professional qualification.

I would like also to express my thankfulness to Prof. Dr. Ramin Yahyapour, who agreed to be the second referee for my PhD thesis.

Especially I would like to say many thanks to Gen Kawamura who was helping me within the whole time period of my PhD. Many thanks also to our other group members Erekle Magradze, Gerhard Rzehorz, Jordi Nadal and Joshua Wyatt Smith. Without their help, support and comments my work would not come to the final phase and I would probably not write this acknowledgement now.

The HappyFace project is a joined collaboration between Karlsruhe Institute of Technology (KIT) and Georg-August-University of Goettingen. Many thanks to Eric Buschmann, Lino Gerlach, Fabian Kukuck, Max Robinson, Christian Wehrberger at Goettingen University and to Stefan Letzelter, Marcus Schmitt, Fred Stober, Guenter Quast, Gregor Vollmer at KIT who had their own contribution in the development of HappyFace project.

Many thanks to Bernadette Tyson, Lucie Hamdi, Gabriela Herbold and Heidi Afshar for all their help with paper work and kind relationship to me.

I want to thank Goettingen International (Vitali Altholz, Janja Kaucic, Bettina Irmer) who provided me the financial support within ALRAKIS Action 2 program.

I would like to say THANKS to all people working in II Institute of Physics, for their help and support.

And finally, I would like to say GREAT THANKS to Jan Schuh, to my beloved parents (Julieta Khalatyan, Rubik Musheghyan), relatives (Norayr Khalatyan, Samvel Khalatyan, Vladimir Khalatyan, Natalia Khalatyan, Natalia Suslova) and to all my friends for their love, support, encouragement. People who are fanning and crossing their fingers for all my achievements and successes.