Journal of Physics: Conference Series **219** (2010) 062057 doi:10.1088/1742-6596/219/6/062057

Site specific monitoring of multiple information systems - the HappyFace Project

Volker Büge, Viktor Mauch, Günter Quast, Armin Scheurer, Artem Trunov

Karlsruhe Institute of Technology, Postfach 6980, 76128 Karlsruhe, Germany

E-mail: viktor.mauch@cern.ch

Abstract. An efficient administration of computing centres requires sophisticated tools for the monitoring of the local infrastructure. Sharing such resources in a grid infrastructure, like the Worldwide LHC Computing Grid (WLCG), goes ahead with a large number of external monitoring systems, offering information on the status of the services and user jobs at a grid site. This huge flood of information from many different sources retards the identification of problems and complicates the local administration. In addition, the web interfaces for the access to the site specific information are often very slow and uncomfortable to use.

A meta-monitoring system which automatically queries the different relevant monitoring systems could provide a fast and comfortable access to all important information for the local administration. It becomes also feasible to easily correlate information from different sources and provides an easy access also for non-expert users. In this paper, we describe the HappyFace Project, a modular software framework for such purpose. It queries existing monitoring sources and processes the results to provide a single point of entrance for information on a grid site and its specific services.

1. Current situation of the monitoring of a grid centre

Current computing centres have to provide a multiplicity of services for different communities and user groups. Most of these services have to be offered with stringent requirements concerning their quality and availability. This includes the quality of data transfers, the performance and reliability of mass storage systems as well as the maintenance of the requested software and service installations. Moreover, the user communities are organised more and more international, relying on multiple computing centres all over the world.

To avoid downtimes in such a complex environment or, if necessary, to ensure that they are as short as possible, a multitude of monitoring applications is needed. They have to provide detailed information about the current status of the system and so to allow a fast detection and thereby a real-time response to problems.

Examples of such a distributed user community are the particle collider experiments in modern High Energy Physics, which have to deal with huge data rates. The experiments of the Large Hadron Collider (LHC) [1] at CERN [2] will create a data output in the order of petabytes per year. Furthermore, it is essential for the scientists all over the world to generate and analyse these data, including corresponding Monte Carlo simulations to compare expected results with real data from the experiments.

The requirements concerning the computing power and storage space led to the decision by the LHC community to build a network of computing centres, which is called Worldwide LHC Computing Grid (WLCG) [3]. Therefore, data storage and processing power from different computing centres have to be organised to provide a working environment for an international collaboration. Each integrated computing centre provides standardised interfaces and software environments for the WLCG so that the work load can be distributed equally on all available sites. The users of the grid are members of Virtual Organisations (VO). These abstract entities group users and resources in a common administrative domain. In the Worldwide LHC Computing Grid, the VOs are associated with the different experiments.

The monitoring of such grid enabled computing centres is an even more complex undertaking. To understand these circumstances, the particularities with respect to the monitoring are described in detail for a centre of the Worldwide LHC Computing Grid.

Such a grid site has very strict obligations concerning the operational availability which is necessary due to the reception and processing of the detector data coming from CERN. There are certain domains to be monitored for a successful operation. First of all, local aspects like the operation of the hardware and a stable infrastructure of the computing centre are the basis. Furthermore, each Virtual Organisation requires an own working environment and software services for stable operation, which have to be monitored, too. On the other hand, there are multiple local grid services, essential for the connectivity to the grid, which are monitored centrally by the WLCG. Moreover, also the functionality of the computing centre for the global VO is monitored. An example for this is the status of data transfers between the CMS grid centres, which is coordinated and monitored by PhEDEx [4]. All these local and global components have to be fully operational to provide a stable working environment for the different VOs. However, the ascertainment of the current status of such a grid site relies on information from many different monitoring sources and is quite challenging.

A look at current monitoring systems of the WLCG reveals numerous difficulties, especially concerning the monitoring of one single grid site and its services. First of all, there are a lot of valuable monitoring applications providing rather unstructured information. For non-experts it is very hard to find out which information is relevant or responsible for a special problem and where it is located. Furthermore, each monitoring application uses its own technology of presenting data values or graphic output. This makes it also difficult to identify correlations between different error sources.

Additionally, the totality of all monitoring systems is uncomfortable to use. A site operator has to check a bunch of different services to access all relevant information. Nearly each monitoring system of the WLCG is designed to store information of all grid sites. Therefore, it is necessary to change the settings in the web interfaces according to own requirements. Another handicap is their high latency. Several monitoring websites, especially complex systems with a database backend, often need more than a minute to submit the settings, query the database and finally to deliver the desired information. By this, the time required for a regular site check is unnecessarily increased.

2. A possibility to improve the situation: meta-monitoring

A system with an adaptable configuration for a special site could automatically query the relevant sources and provide a fast and comfortable access to all required information for the local administration would help to abolish these grievances. Such a meta-monitoring system does not act as an additional source of monitoring information; it would rather create a smart summary from existing ones. A preferable meta-monitoring system can be defined by the properties:

- **flexible**
  The system should be built as general as possible to support all usual system environments.

- **only one web site**
  The final output should be one single website, which shows all requested information from existing sources.
- **up-to-date**
  The complete monitoring information should be renewed within a regular time interval.
- **history functionality**
  It should be possible to use basic history functions like the call of the site status for a specific time or the creation of the status progress for a specific time scale. This functionality can help to find correlations between problems, reported by different monitoring sources at different times.
- **fast access**
  To minimize the load time of the final output it is important to use a system with a very simple architecture, optimised for a fast data access. All information including external plot images have to be stored in a cache or alternatively in a database. The latter should set value on a high performance.
- **comfortable**
  For a highest possible comfort, each result should be accessible with less than three mouse clicks.
- **simple warning system**
  To recognize quickly critical information parts, the framework should provide a simple visualisation system to highlight the results. It can be realised by a display of smileys, arrows or a traffic light logic, which divide the site status in well-defined warning levels.
- **customizable**
  The meta-monitoring framework should provide an easy implementation of customised tests. It should be able to define alert algorithms according to own requirements as well as sharing tests between different instances.

The customers for such a system are site operators or administrators who want to automate site checks and need a quick view on the status of their computer centre taking defined monitoring sources into account. Furthermore, responsible persons from the Virtual Organisations may profit by such systems as it also may summarise information from all important services of their collaboration. Finally, also grid users could get a less detailed view of the system to recognize possible problems of a grid site in case of failure of their jobs.

3. The HappyFace Project

One answer to these premises is the HappyFace Project [5]. This modular software framework is designed to query existing monitoring sources, to process the gathered information and to create an overview of the site status and its services.

3.1. Design

To design the system as flexible as possible and not to be limited by the actual choice of visualisation, acquisition of information and its visualisation are fully decoupled. For each cycle, all collected data as well as the derived results are stored into a database. The visualisation of the stored information then simply goes via a fast query of the local database. Besides offering a dynamic HTML webpage it is also possible to export dedicated information in other formats like XML [6]. Moreover, this decoupling intrinsically offers the requested history functionality.

The framework itself is split up into two parts. The HappyCore provides the basic functionalities of the system like the regular execution of all active modules, access to the database and its initialisation and the composing of the final output of a dynamic web page.

Moreover, it holds basic modules covering functionalities needed by the different tests. In this picture, each test corresponds to a test module with its own configuration and algorithms for the processing of the collected information. The totality of all this specialised test modules builds the second part of the framework.

After the initialisation, the test module collects the requested monitoring information from an external source, processes it and stores the output in the database. Besides the results, additional information like a rating value is also saved. This value is represented by a float number between 0 and 1 where a low value indicates possible problems. This value is used to combine the status of different tests to a global value. To prioritise modules, a weight can be specified. This combination is performed by user defined functions of the HappyCore. In addition, each module creates a PHP [7] fragment for its web output, including the logic for the database query.

The framework finally collects all these PHP fragments and composes the final website. As the information is visualised dynamically, features like the history functionality are intrinsically included.

To provide a most flexible management and optimal customisability, each module is configured by its own two configuration files. The first consists of default settings ensuring the module to be properly executed once activated. The second file gives the possibility to adapt the module for the local particularities of a site.

A very simple example is the HappyCore module "SinglePlot" which provides the functionality to download a plot image from an external website and to store it locally. It prepares the corresponding hyperlink to the database and finally creates the PHP logic to present the desired plot on the monitoring web page. Each test-module which needs such functionality can inherit from "SinglePlot". The only adaption is to set the URL, the test name and its description.

Another basic module is "PhpPlotDashboard" for the download and presentation of statistic plots from the Dashboard monitoring application [8]. The requested dashboard graphic can be obtained by a set of dedicated PHP settings, which are prepared in this module and initialised with settings for the local instance. The test-module inherits from this class and therefore only needs the specification of settings which differ from that default. The advantage of this architecture is that only few adaptations of the "PhpPlotDashboard" configuration are sufficient to migrate the totality of all derived test-modules from one site to another. In case of changes at the monitoring source, only the basic class has to be adapted, not the final modules.

This modular layout intrinsically leads to very small-sized test modules, excludes code doubling and eases therewith considerably the maintenance of the system. It also allows to easily share modules between different sites and to distribute their maintenance.

Figure 1 shows an extract of the inheritance tree. The basic instance is "ModuleBase" which contains the required functionality for each module like database access or the processing of the configuration files. The sub-module "SAM" contains the logic to parse the XML output of the SAM Monitoring Application [9]. The next layer of sub-modules provides algorithms to rate the status of the Computing and Storage Elements. A similar structure is realised with the sub-modules "dCache" and "PoolStatus".

To activate a module, it only has to be specified in the global configuration of HappyFace. To ease the navigation, categories can be specified which hold the final output of each module, keeping the ordering of the configuration file. Please note that only for the webpage, the categories are defined. This allows an easy migration of the module output between categories once needed. For each category, a rating function can be specified used for the calculation of the global category status, based on the status of each member module and its weight. For the monitoring of the German CMS T1 centre GridKA [10], categories for the local infrastructure, the CMS data transfers via the production and debugging instance as well as information on

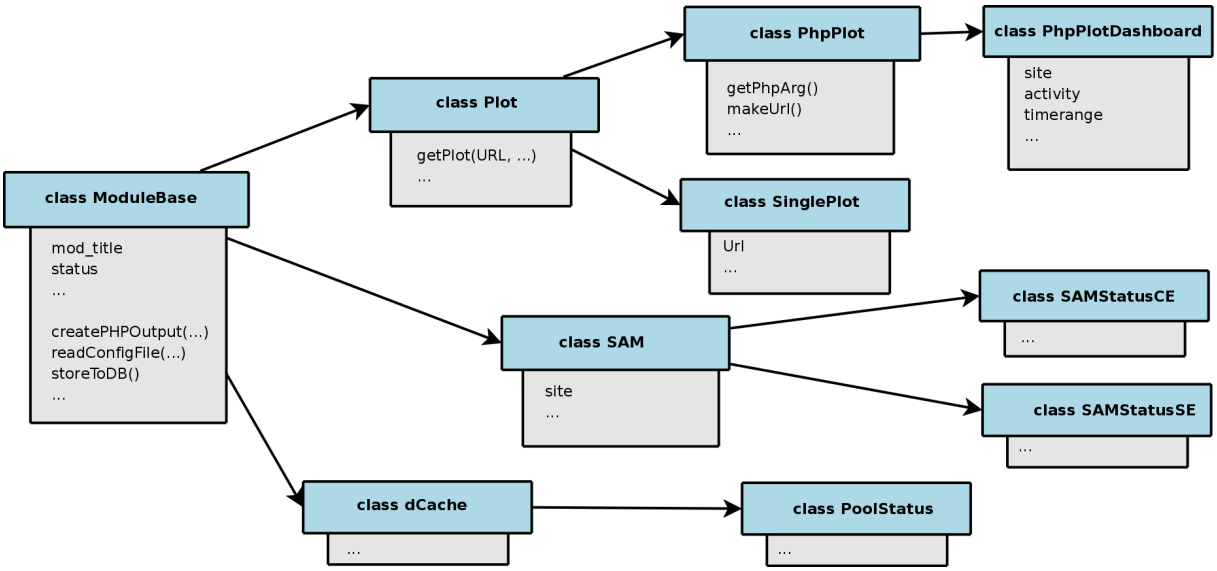

Figure 1. Extract of the inheritance tree of the HappyFace project. The "ModuleBase" provides basic functionalities for each module. The sub-module "SAM" contains the logic to parse the XML output of the SAM Monitoring Application. The next layer of sub-modules provides algorithms to rate the status of the Computing and Storage Elements. A similar structure is realised with the sub-modules "dCache" and "PoolStatus".

the batch and mass storage system are visualised. The rating is performed in a way that the category status is identical to the lowest status of a test module.

A screenshot of the HappyFace Project instance used for the monitoring of GridKA is presented in Figure 2. It shows the output of two test modules, which is arranged in the category "Batch System". The two plots show the efficiency of all CMS jobs during the last 24 houres. The second module monitors the terminated and submitted jobs to this centre via grid means. With the time control bar on the top of the webpage the user is able to query the complete site status for a specific time. This functionality allows to search for status changes back in time and to detect time-dependent correlations between different sources. To make monitoring as comfortable as possible, the complete framework design sets a high value on providing information with a minimal usage of computer input devices. Each test result is achievable in less than three mouse clicks.

3.2. Technical Details of the Implementation

The complete framework is written in the script language Python [11]. To allow an easy implementation of new modules and an extension of the functionality, it has a strictly modular layout, realised using the concept of inheritance.

For the configuration, the standard Python ConfigParser is used. Hereby, each module has access to the configuration of all classes from which it is inherited. If a configuration variable is set several times, the one of the last occurrence in the inheritance tree is chosen.

The decoupling of data acquisition, processing and visualisation is realised with a SQLite [12] database, in which all kinds of ASCII data is stored. To keep the database as fast as possible, binary files like plot images are stored in the local file system and only the path to the local copy of the file is accessible via the database. The final webpage consists of the PHP fragments from the different modules, which contain the complete logic to query the database depending on the point in time requested by the user. Once a time is specified, the timestamp in the

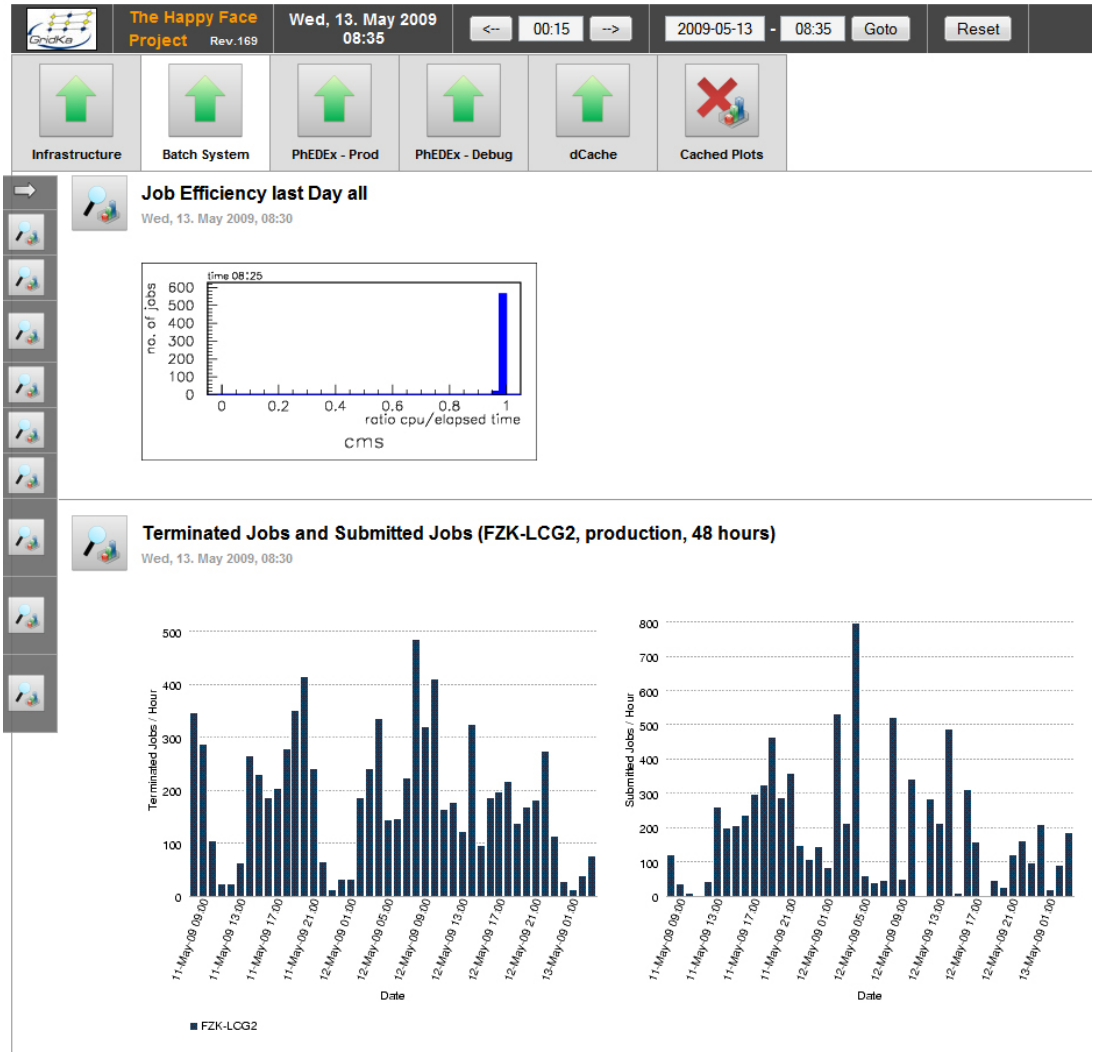

Figure 2. Screenshot of the HappyFace Project instance used for the monitoring of the German CMS T1 centre, GridKA. It shows the output of two test modules, which is arranged in the category "Batch System". The two plots show the efficiency of all CMS jobs during the last 24 hours. The second module monitors both terminated and submitted jobs to this centre via grid means. With the time control bar on the top of the webpage the user is able to query the complete site status at a specific time.

database closest to the given one is calculated and the corresponding information is displayed. Per default, the webpage displays the latest information and updates every five minutes. In Figure 3 the workflow of the framework is illustrated.

Please note that the current web output based on PHP has been chosen in order to minimize the requirements on the web server hosting the HappyFace Project. It is rather easy to add additional output webpages based on other technologies.

All modules are executed at the same time in a multi-threading environment. In addition to the economy of time this procedure has the advantage that modules, exceeding a global timeout setting, can be terminated by the system. This is an important issue to handle the case that an external source may have temporary problems to provide the requested data.

Other possible internal errors of the modules usually concern inconsistent or missing

information. For these cases, error handling routines catch the failure and display an error message in the final output.

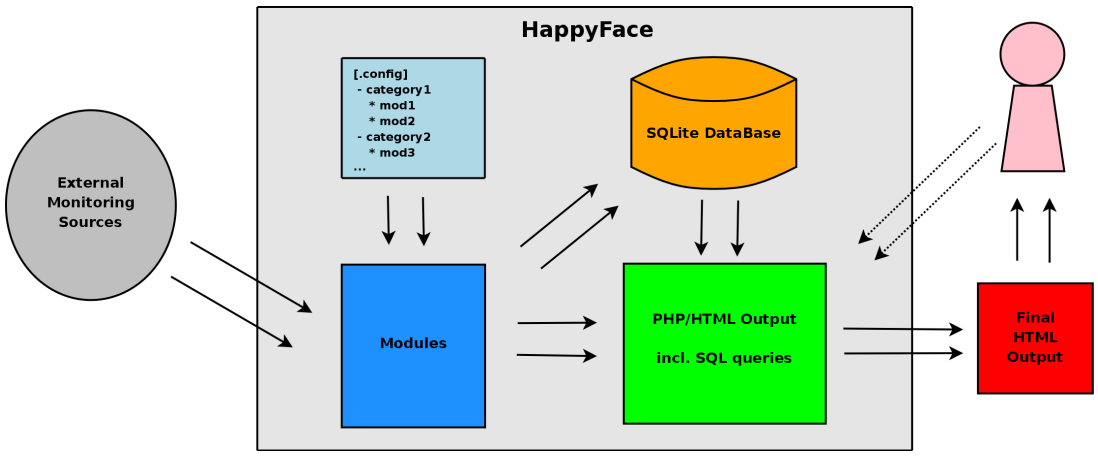

Figure 3. Workflow of the framework. Each active module collects the requested external data, process and rates it. The relevant results are stored in the database. An output fragment is provided by each module, later being composed to the final output webpage. This file contains the complete SQL query logic, which is used by visiting the website and request the status concerning to a specific timestamp.

3.2.1. Requirements The HappyFace Project requires only very basic programs and libraries commonly available under GNU/Linux, thus it can easily run on a desktop workstation. A mechanism for an execution in periodic time intervals, such as cronjobs, as well as Python (v2.5.2 or newer) are required. Furthermore, the database engine SQLite3 and the Python package SQLObject [13] provide the database functionality. The web server has to support the script language PHP (v5.x).

4. Related Work

Besides this approach there are further monitoring systems providing availability and status information of the grid sites and their services. However the overall aim is another in comparison with the HappyFace Project. Most of them, like the SiteView GridMap Monitoring Tool [14], try to provide a grid-wide monitoring information centre for all existing sites. The main difference of these, compared with the HappyFace Project, is the centralized maintenance. People using such services have no direct influence concerning rating algorithms, used monitoring information and display output. Furthermore centralized tools have a natural limitation on accessing monitoring information.

In contrast, a local monitoring instance of a grid site can be configured according to desire. Local information, which is perhaps not released for the publicity, can also be taken into account for a more sophisticated status rating. The responsible site administrators can realize their own analyze logic, alert warning levels and display output. The HappyFace Project fulfils exactly this concept. The framework allows using existing modules and provides the possibility to build and use new modules if required. Each aspect of a HappyFace monitoring instance from the choice of relevant information via the rating through to the display output is configurable and under the control of the concerning administrator who use this tool.

5. Conclusion & Outlook

An efficient administration of computing centres requires sophisticated tools for the monitoring of the local infrastructure. Especially if the centre is connected to the grid, a multitude of internal and external monitoring sources has to be checked to get reliable information on the status of a grid site. However, these monitoring sources have limitations with respect to their response time and the needed information is often not clearly accessible.

One possibility to improve that situation for the site administrators is a meta-monitoring, which automatically queries the relevant sources and provide a fast and comfortable access to all required information. It is not an additional source of information but offers a smart summary of existing ones.

HappyFace fulfils all requirements that have been defined for a preferable meta-monitoring system. Due to the decoupling of the acquisition and visualisation of the information, the system is very flexible and can be adapted for many different monitoring purposes. Moreover, due to its modular structure, the development of new modules and their maintenance can be shared between different instances.

The modular structure of HappyFace fits perfectly to the fact that most grid centres have to control the same monitoring sources, only with a slightly changed PHP request or xml data. Here, it is often sufficient to configure one parameter in the HappyCore modules to adapt multiple existing test modules for the local particularities. The same is valid once the format of the output of a monitoring source changes as only a modification of the corresponding core module is necessary. Once adapted, it can be published and all other instances can update.

To benefit most from this plug-in system, HappyFace is maintained in a subversion repository. Each available module comes with a default configuration that works at every site out of the box once activated at a site. The adaptation of a module for the particularities of a site takes place in a local configuration file. As soon as a fix in one of the basic modules is available, the instance can be updated without loosing the local configuration. The same is valid for new modules.

The HappyFace Project is already used for the monitoring of the CMS activities at the German CMS T1 Centre GridKA during several computing challenges. Especially the clear arrangement of the required information helped non-expert shift crews to monitor such a complex system. Moreover, HappyFace instances are used for the monitoring of the WLCG centres in Aachen, Göttingen and Hamburg/DESY by local groups of the ATLAS or CMS experiment.

Besides the improvement of existing and the development of new modules, the XML export of the category status as well as other key information will become available in near future. Having defined categories and rating functions, such an XML output can be used to summarise the status of several computing centres allowing a central shift monitoring multiple centres. Once a failure is detected in one of the categories of a centre, the local HappyFace instance can then give more detailed information.

Having these functionalities implemented, HappyFace contributes to an efficient monitoring application of complex grid centres and eases the discovery of failures, leading to a better reliability and availability for the user communities.

Acknowledgment

We thank Stefan Birkholz, Friederike Nowak and Philip Sauerland for the fruitful discussion, testing of the recent developments and the maintenance of some modules.

Furthermore, we acknowledge the financial support of the Bundesministerium für Bildung und Forschung BMBF as well as the Helmholtz Alliance 'Physics at the Terascale'.

References

[1] Large Hadron Collider, http://lhc.web.cern.ch/lhc/
[2] Organisation Européene pour la Recherche Nucléaire (CERN), http://www.cern.ch/
[3] Worldwide LHC Computing Grid, http://lcg.web.cern.ch/lcg/
[4] PhEDEx, http://cmsweb.cern.ch/phedex/
[5] HappyFace Project, https://ekptrac.physik.uni-karlsruhe.de/HappyFace
[6] XML (Extensible Markup Language), http://www.w3.org/XML/
[7] PHP, http://www.php.net/
[8] Arda Dashboard, http://dashboard.cern.ch/
[9] Site Availability Monitoring (SAM), http://lxarda16.cern.ch/dashboard/request.py/samvisualization
[10] GridKa, http://grid.fzk.de/
[11] Python, http://python.org/
[12] SQLite, http://sqlite.org/
[13] SQLObject, http://sqlobject.org/
[14] SiteView, http://dashb-siteview.cern.ch/