Using widgets to monitor the LHC experiments

2011 J. Phys.: Conf. Ser. 331 072015

(http://iopscience.iop.org/1742-6596/331/7/072015)

Journal of Physics: Conference Series **331** (2011) 072015 doi:10.1088/1742-6596/331/7/072015

Using widgets to monitor the LHC experiments

I González Caballero[1] and S Sarkar[2]

[1]Universidad de Oviedo, Asturias, Spain
[2]SNS, Pisa, Italy

E-mail: Isidro.Gonzalez.Caballero@cern.ch

Abstract. The complexity of the LHC experiments requires monitoring systems to verify the correct functioning of different sub-systems and to allow operators to quickly spot problems and issues that may cause loss of information and data. Due to the distributed nature of the collaborations and the different technologies involved, the information data that need to be correlated is usually spread over several databases, web pages and monitoring systems. On the other hand, although the complete set of monitorable aspects is known and fixed, the subset that each person needs to monitor is often different for each individual. Therefore, building a unique monitoring tool that suits every single collaborator becomes close to impossible. A modular approach with a set of customizable widgets, small autonomous portions of HTML and JavaScript, that can be aggregated to form private or public monitoring web pages can be a scalable and robust solution, where the information can be provided by a simple and thin set of web services. Among the different widget development toolkits available today, we have chosen the open project UWA (Unified Widget API) because of its portability to the most popular widget platforms (including iGoogle, Netvibes and Apple Dashboard). As an example, we show how this technology is currently being used to monitor parts of the CMS Computing project.

1. Monitoring the world

The experiments operating at the Large Hadron Collider (LHC) and, more generally, any experiment or complex system needs to continuously monitor all its hardware and software components. This is done by building databases that store the relevant figures and states of the different sub-components. Web interfaces and dedicated applications are built on top of this information to construct plots and tables reflecting the values and trends of the relevant quantities. Eventually all these high level monitoring objects are put together according to their affinity in web monitoring portals. Typically, the LHC experiment monitoring information is scattered over several databases, web pages, and some other dedicated systems. They might be directly accessed through a summary web page followed by a few links, or through well defined web services.

This project was launched in the context of the computing activities at the Compact Muon Solenoid (CMS [1]) experiment where the information required to efficiently monitor the status of this extensive system is distributed over the PhEDEx[2] web page and web services (data placement and transfers), DBS[3] web service (data location), several projects in the CMS version of the Dashboard project for LHC experiments [4] (workflows, site status and readiness), FroNTier/Squid[5] web page (conditions database), several local monitoring portals at sites and a few other related pages.

It was soon understood that the different monitoring actors in CMS (users, experiment supervisors, shifters, service operators and system administrators at the different sites, to mention a few) are interested in different types of information. Only portions of the overall data might be common to several of these actors. To make things a bit more complicated the responsibilities are not necessarily split in the same way at every site. Moreover, those responsibilities might change over time as the systems become better understood and, therefore, more stable, and as new services are added in. Additionally, people looking after a site or a service know better its strong and weak points so they need to focus their monitoring efforts on those aspects that are unstable or more prone to failure.

As a consequence, building a unique monitoring tool that suits every single actor becomes close to impossible. However, several solutions can be offered in order to confront this problem.

In the simplest approach each actor would collect a set of interesting URLs, possibly in a simple web page, to browse them in a sort of infinite loop. Apart from the fact that continuously navigating through a large number of links is rather uncomfortable, the pages one looks at might include not only the plots and metrics the actor is interested in, but also several irrelevant ones, making the activity quite inefficient. Keeping all the URLs up to date in an evolving environment is also something that might become difficult if the number of monitoring actors is big.

Fortunately most systems that need to be monitored are web based so they can be thought of as web services providing monitoring data through an API. This API might be as simple as a set of URLs to the interesting plots or somewhat more complicated (web based data queries). Therefore, another possible solution to deal with the monitoring tasks would be to construct personal or institutional monitoring web pages aggregating the relevant information for each actor. The main drawback is that each individual building such web pages is required to master at least HTML coding and most probably a set of other technologies and formats like JavaScript, XML, JSON etc. If the number of different types of actors is big, the number of people that would need to have such skills becomes too big to be practical. Moreover, any API changes would require each page to be fixed accordingly.

Finally, one can think of a modular approach in which the information provided by the web services mentioned above is served in small pieces to the monitoring actors so that it can be easily aggregated to form a suitable monitoring dashboards. Web widgets are small applications that can be installed and executed within a web page by an end user. Through the different widget platforms available several widgets can be put together to form private or public monitoring web pages. Although widgets are single units of information, they are not necessarily simple and they can perform complicated tasks like combining information from different sources. This modular approach allows each actor to easily customize the monitoring view to the needs of his duties. In the event of API changes a fix in the widget code by its developer will automatically propagate to all the consumers. The number of people with special skills is significantly reduced. Another advantage of a widget based monitoring compared to placing several links on a single page is that refresh times can be set independently and may be optimised to load the server-side minimally while in case of a page with many links, all get updated at the same frequency.

In the context of our project we foresee three use cases that can benefit from using widgets:

- Individual users or experts may build their own monitoring display with the set of plots and tables that better shows the status of the systems they needs to look after
- Each site may build a monitoring web page with only the external and internal information that may be relevant to the rest of the collaboration, should they want to know the status of their particular cluster.
- Physics groups may offer a web page that aggregates the relevant information about the status of their associated sites.

It might be mentioned that the widget technology is not to be seen as a replacement to the

current various dedicated and quite complete monitoring systems out there, but it can be better thought of as an orthogonal complement to them. A monitoring system based on widgets needs itself web based information providers.

2. Universal Widget API

There are several widget platforms available. Some of them are independent of the operating system they are run in by being completely web based like Netvibes[1] and iGoogle[2] or by depending on a cross platform web browser like Opera Widgets[3]. Some others are specific to a given operating system: Windows Sideshow[4] or Yahoo! Widgets[5] for Windows based systems, Apple Dashboard[6] for Mac OS systems (including, for example, iPhones). All the platforms provide a well defined API to build the widgets and an easy way to add them in a coherent view. Unfortunately the widgets developed with most of those APIs are restricted to a single platform and are incompatible with the rest.

All the widget platforms provide similar functionality in terms of things like flexible and easy mechanisms to add widgets (usually through a dedicated portal), support for tabs and different layouts, with very little difference among them. On the other hand most of the APIs support not too different capabilities like automatic view refresh, remote content management, widget styling and user settings support.

Given the situation described above one could select one of the platforms with its associated API and restrict the implementation to the chosen option. While this would avoid the usual caveats of cross-platform development, it would force all the actors to conform to it. This is clearly not desirable. The Unified Widget API (UWA) developed by Netvibes provides a solution to all those concerns.

Our selection of the UWA is based in the following reasons:

- It is an open platform with a well defined API and part of it is already Open Source code. Because of it more degrees of customisation are possible through forking the main project. For example, in case more platforms need to be supported or extra functionality needs to be added to the widget server, the core code can be easily modified.
- Nothing else except XHTML and JavaScript is needed to build a widget.
- The widgets developed with UWA can, in principle, be used through a variety of widget platforms including the most usual ones: Netvibes, iGoogle, Windows Sideshow, Apple Dashboard, Opera Widgets or Yahoo! Widgets. This leaves the monitoring actors the choice on the widget platform allowing them to use the technology they feel more comfortable with. Adding UWA widgets to a web page or even a blog is rather easy. A couple of examples of monitoring views in different platforms are shown in Figure 1.
- It provides a huge collection of existing widgets through the Netvibes Ecosystem[7], some of them very useful to complement any monitoring page. Among others one can find the usual task lists, calendars, clocks, RSS feed aggregators, etc.

All the widgets can be exposed through the already mentioned Netvibes Ecosystem providing a central portal to access them independently of the platform they will be added to. Though it is not a key feature, it might be interesting in some context to point out that UWA has built-in

[1] http://www.netvibes.com
[2] http://www.google.com/ig
[3] http://widgets.opera.com/
[4] http://windows.microsoft.com/en-US/windows/downloads/personalize/gadgets
[5] http://widgets.yahoo.com/
[6] http://www.apple.com/downloads/dashboard/
[7] http://eco.netvibes.com

support for localization. There is plenty of documentation available for developers including several support fora.

Nevertheless, we have found some aspects of UWA that could be improved. The main limitations are the lack of support for modular programming and for authentication through certificates. The first point complicates the design and development of a coherent set of widgets. The second one makes it difficult to access the information behind web services that support authentication only by this mechanism.

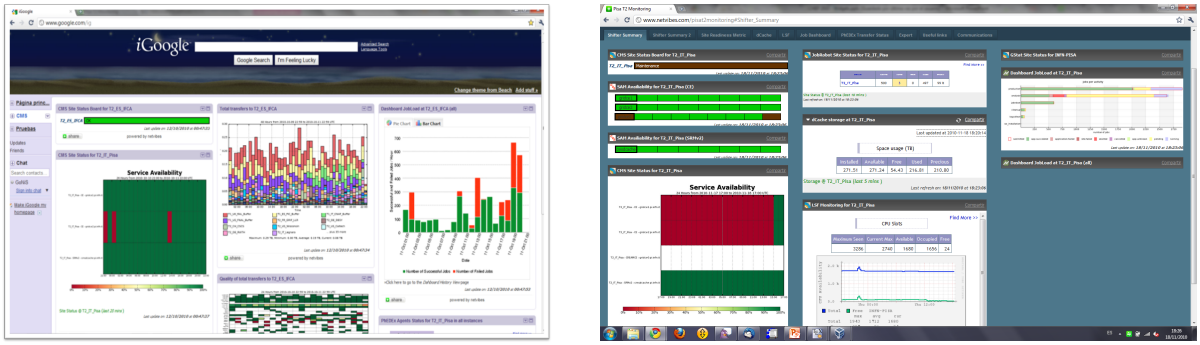

Figure 1. Monitoring web pages built using UWA widgets developed in the context of the CMS Computing project by aggregating them to the iGoogle (left) and Netvibes (left) platforms

3. Basic structure of a UWA widget

All the UWA widget have a pretty simple structure[8] that must be contained on a single file. This file is coded in standard XHTML, except for the preferences section, where a specific set of elements are used. The XHTML code is entirely based on Web-standards: XHTML/XML, JavaScript/Ajax and CSS.

The structure of a UWA widget can be schematically represented as shown in Figure 2. The code can be divided into three main blocks:

- A set of standard XHTML headers.
- The core block where the behaviour and aspect of the widget is defined through a set of special tags, a few optional styling lines in CSS and some JavaScript code.
- The HTML skeleton of the widget that might be altered by the code inside the previous block.

The core section can itself be considered as made of the following four different sub-blocks:

- Meta Information: Including author name and email, short description, widget and API version, thumbnail, keywords, etc. This information can be distilled by other tools like the already mentioned Netvibes Ecosystem to classify and expose the widget to the user community.
- Preferences: It defines how the widget users interact with it, i.e. the set of options customisable by the user. Preferences are based on a UWA-specific XML element (`widget:preferences`) and a special tag (`preferences`). There are several types of preferences available including checkboxes, list boxes, text fields or passwords. These preferences are stored remotely, and retrieved every time the widget is loaded.

[8] A more detailed and updated description of the structure and internals of UWA widgets might be found in the official documentation kept in `http://dev.netvibes.com`.

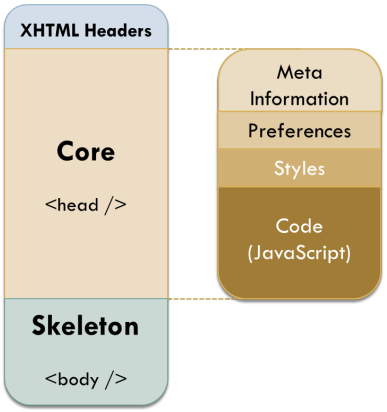

Figure 2. Schematic view representing the structure of the code for an UWA widget.

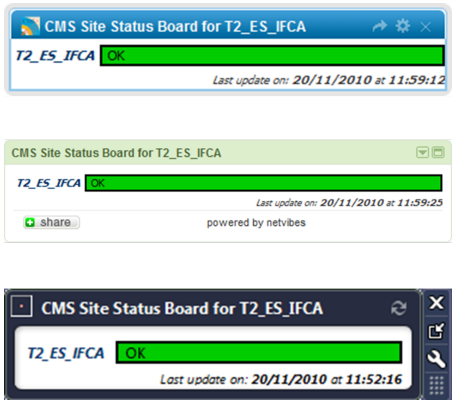

Figure 3. Three versions (in Netvibes, IGoogle and Windows Sideshow) of the widget showing the status of a given site according to the CMS SSB metrics.

- Styles: Defined using CSS.
- Action: This is the JavaScript code that grabs the information from the preferences section and manipulates it to build the presentation layer. The UWA provides mechanisms to find out the preferences, retrieve information in various formats from the web (services), as well as hooks to interact with the HTML skeleton in the next block.

4. Widgets for CMS Computing monitoring

We have developed a set of widgets to monitor the status and activities of the CMS Computing sites. Some of the widgets have been extended to include some extra useful functionality, like support for updates and debug output. Some degree of standard look and feel is achieved by imposing a similar structure and sharing code (HTML and JavaScript) wherever possible. The widgets showing monitoring information are available in the following areas:

- Data location: There are widgets to monitor the status of the PhEDEx services at each site (status of the different agents and transfer links), as well as its performance by showing not only the rate, volume and quality of the global data transfers (see Figure 4), but also the amount of data for a given dataset that has been transferred to a site. The amount of space used by the different types of data according to its association group is also provided by one of the widgets.
- Work flows: The widgets for the interactive and historic views provided by the CMS Dashboard allows the users to follow, in their monitoring views, the load in terms of jobs at a given CMS associated site. An example is shown in Figure 5.
- Site Status: The results of the SAM tests for the relevant services at any CMS site as well as the results of other tests and metrics performed over CMS tiers (JobRobot, CMS Site Status Board, Site Readiness, etc.) can be followed through specific widgets. We also provide a specific widget for the FroNTieR service.
- Local services monitoring: We have developed widgets to monitor the details of some local services at the sites like the dCache storage element and the LSF batch system. There is also a widget that correlates the dashboard information with information directly coming from the local batch system.

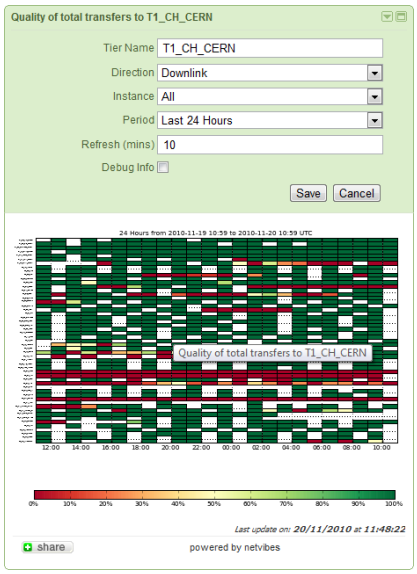

Figure 4. Widget showing the quality of the transfers associated with a given site according to the information provided by PhEDEx.

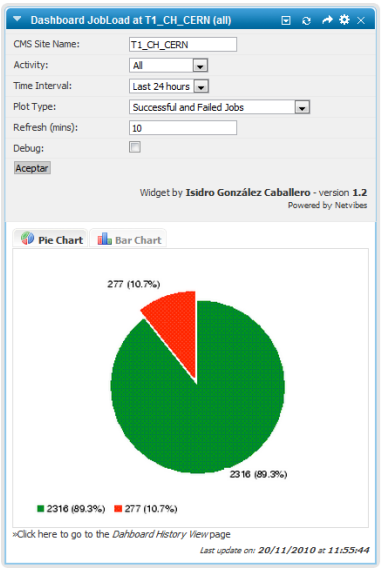

Figure 5. Widget showing several plots with information on the state of jobs at a given site extracted from the Dashboard project for the LHC experiments.

5. Conclusions

The widgets technology enables a different way of visualizing the same monitoring data whenever there are good web data services available. The modular approach to serve information provides a framework with a high level of flexibility and customisability which is very difficult to get with other solutions. It solves the problems arising from trying to implement a unique monitoring tool that should suit very different and numerous use cases. Changes or improvements in the data service interfaces can be quickly propagated to the consumers.

Widgets based on the Unified Widget API can be used in a great variety of platforms including the most popular ones. Developing new widgets using the UWA is rather easy and only the knowledge of a bit of XHTML and JavaScript is required.

Widgets are being efficiently used to monitor the CMS Computing services. Around 10 CMS Tier-2 sites are actively exposing monitoring information through widgets. Some of them have even developed shift instructions based on those pages. There are also a few CMS physics groups using public web web pages built with widgets to check the status of sites they are associated with. There are individuals with different responsibilities at the sites that are actively using widgets in their desktops or dashboards to construct the views that better suits their needs.

References

[1] The CMS Collaboration et al 2008 *JINST* **3** S08004
[2] Rehn J et al. 2006 *Proc. Conf. Computing in High Energy Physics 2006 (CHEP2006)* vol. 2, ed S Banerjee (India: Macmillan) p 1027
[3] Afaq A, Dolgert A, Guo Y, Jones C, Kosyakov S, Kuznetsov V, Lueking L, Riley D and Sekhri V 2007 *The CMS dataset bookkeeping service* Proc. Conf. Computing in High Energy Physics 2007 (CHEP07) (Victoria:IOP)
[4] Andreeva J, Boehm M, Gaidioz B, Karavakis E, Kokoszkiewicz L, Lanciotti E, Maier G, Ollivier W, Rocha R, Saiz P et al. 2010 *J. Grid Comput.* **8** 323-339
[5] Lueking L, Kosyakov, S, Kowalkowski J B, Litvintsev D O, Paterno M, White S, Blumenfeld B, Maksimovic P and Mathis M 2004 *Proc. Conf. Computing in High Energy Physics 2004* (Interlaken, Switzerland) p 669