Fast access to the CMS detector condition data employing HTML5 technologies

Giuseppe Antonio Pierro¹, Francesca Cavallari², Salvatore Di Guida², Vincenzo Innocente²

¹ INFN-Bari - Bari University, Via Orabona 4, Bari 70126, Italy
² CERN Geneva 23, CH-1211, Switzerland

E-mail: antonio.pierro@gmail.com

Abstract. This paper focuses on using HTML version 5 (HTML5) for accessing condition data for the CMS experiment, evaluating the benefits and risks posed by the use of this technology. According to the authors of HTML5, this technology attempts to solve issues found in previous iterations of HTML and addresses the needs of web applications, an area previously not adequately covered by HTML.

We demonstrate that employing HTML5 brings important benefits in terms of access performance to the CMS condition data. The combined use of web storage and web sockets allows increasing the performance and reducing the costs in term of computation power, memory usage and network bandwidth for client and server. Above all, the web workers allow creating different scripts that can be executed using multi-thread mode, exploiting multi-core microprocessors. Web workers have been employed in order to substantially decrease the web page rendering time to display the condition data stored in the CMS condition database.

1. Introduction

The web application language has been so far stranded on HTML version 4 (HTML4), which is more than 10 years old and which is not adequate for web applications. The web site of most major private companies arranged to overcome the limitations related to HTML4 obsolescence by exploiting a proprietary plug-in-based on Rich Internet Applications (RIA) technologies such as Adobe Flash, Microsoft Silverlight, Apache Pivot and Sun JavaFX. Since policies at CERN, as well as in other HEP laboratories, encourage the use of open source software, the CMS off-line database group has been so far implementing code based on JavaScript, PHP or Java in order to deploy more interactive and dynamical web applications; still, it has been hard to deliver some features with the available technologies.

HTML5 is by now widely regarded as the next standard for web applications. Indeed, it will be, de facto, employed by Internet Explorer, Firefox, Safari, Google Chrome and other browsers for web site rendering. Therefore, our group has decided to start making use of HTML5 for accessing and managing the huge-size of CMS conditions data over the World Wide Web (WWW). This paper explores techniques used in recent years to address the non-optimal connectivity of the RIA, and describes the implementation of some Application Programming Interfaces (APIs) provided by HTML5 specifications, which enhance the features of RIA.

The structure of the paper is as follows. Section 2 contains brief details on the meta-data used for handling CMS conditions data. In Section 3 we motivate our reasons to deploy a web-
based GUI interface instead of a desktop application, and specifically our choice to exploit new HTML5 web technologies, even if they are still currently under development. In the following three sections, we describe the APIs currently under development that have been exploited in the context of CMS conditions data, providing some examples directly related to the web applications. The HTML5 APIs that we describe in detail are: web workers, web storage and web socket. Results and conclusions are given in the last section.

2. CMS condition data elements
In order to setup the CMS detector and to reconstruct physical events, physicists use a large amount of non-event data, also called condition data that describe the behaviour of the various CMS detector components. The condition data contain many different specific data items, such as mapping to electronic channels of the Resistive Plate Chamber (RPC), alignment constants of the Silicon Strip (SiStrip) detector, and calibration of electronic circuits of the Electromagnetic Calorimeter (ECAL). These condition data may change over time to reflect changes in configuration or as a more precise understanding of the detector’s behaviour become available. For these reasons each conditions data object is identified by:

- a unique human-readable name (TAG) which includes
  - the CMS sub-detector element, e.g. ECAL, RPC, SiStrip.
  - the type or nature of the data, e.g. pedestal, geometry, temperature.
  - the version of the CMS Software (CMSSW), e.g. 31X, 34X.
  - a progressive version number, e.g. v1, v2, which means the condition data have been recalculated so to make use of a wider statistics.

- The Interval of Validity (IOV), a validity range with a start time (since) and an end time (till).

The actual values of the physical observables associated to a condition data object are referred to as the ”payload” of that conditions data object.

The web application used by physicists to discover TAGs, IOVs and payloads of each set of conditions data is called Payload Inspector. From this web application point of view, the entry point for the visualization of the conditions data stored in the condition database is the CMS condition data Account Name, that includes the sub-detector and the CMSSW release employed to produce the condition data value.

3. Motivation
For the CMS experiment, there are significant data storage requirements for non-event data. A large amount of condition data is required for physics analysis, including: calibration data, alignment and survey data, as well as detector description data [2], [3]. Such data needs to be accessible by the whole collaboration. Before discussing which technology we apply to provide the condition data to CMS experiment community we focus on both the advantages and the disadvantages of the different software applications (desktop vs web application) and different markup language for web pages (HTML4 vs HTML5). In the following, we aim justify the choices of our implementation, a web application using the latest web technologies.

3.1. Desktop vs web application
There are two ways of implementing a user interface, either using a desktop application or a web application. Both methods offer advantages and disadvantages. Following some of the ISO 9126 and most recent ISO 2500 software quality characteristics, we can compare the two approaches in terms of quality requirements:
• Maintainability, Portability and Installability. Web based applications need to be installed only once on the server; while desktop applications need to be installed separately on each computer.
• Changeability. Updating applications would need to be done on every single computer, which is not the case with web applications.
• Usability and Functionality. Desktop applications are confined to a physical location and hence have usability constraints.
• Interoperability. Using a web browser to access the application provides cross-platform compatibility.
• Security. Web applications are exposed to more security risks than desktop applications. The web applications, as they are open to a large number of users in the internet community, are exposed to a major threat from unauthorized access, modification, or destruction.

The CMS condition database developers will take advantage of building web application since:
• the users employ a variety of different operating systems (OS) to access the condition data, and creating different applications for different operating systems is quite hard.
• the software upgrades are frequent (for example, to accommodate various users requests and to improve performance).

Moreover, the CMS condition database users will profit from the access of the application from any location using the Internet without need to install any additional software.

3.2. HTML5 vs HTML4
At the time of this paper, HTML4 has been in service for more than a decade. In this time the WWW has evolved. New and innovative websites are being created every day pushing the boundaries of HTML in every direction and really pointing out the need to upgrade the core markup language of the web to the next major revision, namely HTML5.

In this paper we focus on three APIs from HTML5, web workers, web storage, and web sockets discussing the benefits that can be gained from using them, in the context of accessing and monitoring the CMS condition database.

4. Web Workers
The next round of innovative applications will require more computing power. The hardware vendors are responding, not with faster Central Processing Units (CPUs), but with multi-cores and many-cores, often combined with Graphics Processing Unit (GPUs) and other types of accelerators [4].

In order to exploit the capabilities of these new architectures, the programming language community has responded with new tools for parallel programming. All the traditional application-programming languages (C, C++, Java, etc.) are increasing support for parallel computations in multiple different ways through extensions, libraries, virtual execution environments, automated parallelization tools, parallel programming patterns, and so forth. Surprisingly, while these approaches have been at least partially successful in their own application domains, their adoption into the web-oriented programming environments and scripting programming languages, used by a larger and larger number of developers, had never been addressed.

Only recently, thanks to the most recent developments such as HTML5, the API Web Workers allow parallel processing of web client scripts. In detail, HTML5 Web Workers provide background-processing capabilities to web applications and typically run on separate threads so that JavaScript applications using such APIs can take advantage of multi-core CPUs.
A numerical computation job was run to obtain a reliable result about the contribution of web workers, removing any dependency on external agents such as loading operations from network external resources. Such a numerical computation has been performed on Safari 5.0 run over an Intel Core i7-860 (2.8 GHz, quad core, Hyper Threading) processor. Table 1 shows the results obtained by running a JavaScript algorithm [7]. The test computes $2^{2048 Mod(97777)}$ by using the Chinese Remainder Theorem [5] and the map/reduce problem to merge all threads before computing the final result [7].

| Number of workers | Average time | Average Time (normalized) |
|-------------------|--------------|--------------------------|
| 1 Worker          | 20836 ms     | 100%                     |
| 2 Workers         | 10920 ms     | 52%                      |
| 4 Workers         | 6054 ms      | 29%                      |
| 8 Workers         | 3338 ms      | 16%                      |
| 16 Workers        | 3325 ms      | 16%                      |

Table 1: Results of JavaScript benchmark processes using different numbers of Web Workers: 1,2,4,8 and 16.

One way Workers can be useful, in the context of accessing the CMS detector condition data, is to allow the code to perform processor-intensive calculations without blocking the user interface thread. In our context, each CMS sub-detector may have over two hundred TAGs, each having a thousand IOVs for which you can get payload in different formats: XML, plot, trend plot. Executing a single-threaded JavaScript code will yield a larger latency, and, at the same time, the user will not be able to use the GUI while the page is rendering the full content of a specific CMS condition data Account Name. In order to avoid this bottleneck, we allow the rendering work of the HTML page to be divided among each processor through different JavaScript threads. For example, in the case of ECAL CMS condition data Account Name, the page rendering of HTML elements is divided in five JavaScript threads, each run separately in series or/and in parallel according to the number of cores used.

The benchmark results comparing web workers with single-thread JavaScript for the Payload Inspector are presented in Figure 1. The web page rendering timeline depends on several factors including:

- loading time, directly related to the website components (blue timeline),

Figure 1: Two snapshots from Safari developer tool [6].
• the time interval for running the scripts (orange timeline),
• rendering time to display the web page (purple timeline),
• web browser and, most of all, parallel processes running on workstation and connection speed.

By using **Web Workers** it has been possible to reduce the JavaScript execution time (orange timeline) by over 20% and the rendering process time (purple timeline) by over 52%. The test has been performed with a 1.86 GHz Intel Core 2 Duo processor.

5. Web Storage

The **Web Storage** specification defines an API for persistent data storage in the web applications without the need for server side storage. It means that we can store data that, in case the end-users do not clear their browser data off, will be made available at a later time for the subsequent use of the client.

One example of web storage, useful in the context of accessing the CMS detector condition data, is to cache the tag containing the full sequence of IOVs. Since in the condition database, for each CMS condition data Account Name, most of the data does not change and can be stored on the client disk to make it available to the user without having to re-fetch it from the server. This method has several advantages:

• save network bandwidth and load on the server: every time you load a page that has some cached elements, the browser will poll the server to check if the cached file has been updated; if not, then it will not download it. By doing so, the load on the server is considerably reduced, and the bandwidth usage will be also minimized.
• reduce time response by over 90%: files that are cached locally will load much faster.

On the bottom, the Figures 2a, 2b show the page resources by size for **Payload Inspector** using HTML4 and the size for **Payload Inspector** using Web Storage, respectively. By using Web Storage we reduce network traffic by over 98% (772KB using web storage vs 17.18MB using HTML4). The test was performed, for both web applications, from the **CERN General Public Network** (CERN-GPN).

![Figure 2a: Page resources by size for Payload Inspector using HTML4.](image1)

![Figure 2b: Page resources by size for Payload Inspector using Web Storage.](image2)

6. The Web Sockets API

Using HTML4, **polling operation** is the default way of getting data from the server. A client sends request to the server, and the latter responds providing the required data. The server cannot send any data to the client by itself without the client having asked for it. This is called transitory connection, when only one side can request for data. With new web emerging technologies, more functionalities from websites are demanded, such as full communication between server and clients, or allowing server requests to the client. Such functionality was impossible to achieve in http protocol without additional workarounds, because it was not designed to work in such a way. Over the years there has been various techniques developed to simulate server
pushing the data to the client, like continuously polling in constant time intervals, which brings a lot of unnecessary internet traffic, and long polling - when the client polls the server but the server responds only when new data arrives or after a suitable time amount, so reducing internet traffic. The complications of handling connections between client and server are avoided using the Web Sockets API. Thanks to this API a continuous connection between the server and the client is created and it is not automatically closed after the server sends the data back to the client, thus allowing bidirectional communication between client and server. No headers are exchanged once the single connection has been established, server polling for checking if new data are found can be avoided. These features reduce both internet traffic and server load.

In the context of the CMS condition database we use the Web Sockets API in order to keep condition metadata updated. In detail, it works in this way: a watchdog, running on the server, looks for data changing by checking just the IOV size. If the IOV size has been changed since the previous report, the watchdog performs the query to the condition database. The watchdog sends to the client just the new IOV entries for each TAG: a few bytes are enough to keep the client updated, so that users can see the change without waiting for page refreshing.

7. Conclusion

Employing HTML version 5 technologies has proven to bring major benefits in terms of performance in accessing the CMS condition data. We have demonstrated with some examples that the combined use of web storage and web sockets allows increasing the performance and reducing the cost in term of computational power, memory usage and network bandwidth for client and server. Above all, the web workers allow creating different scripts that can be executed using multi-thread mode, exploiting modern multi-core microprocessors. Web workers have been employed in order to substantially decrease the web page rendering time to display the condition data stored in the CMS condition database. Despite the significant benefits coming from these three HTML5 APIs, the web security experts warn about possible security issues related to this new technology, especially web sockets, currently under development. For this reason in this work, we have evaluated the use of this technology without putting into production the web sockets, waiting that it passes the compatibility tests about security issues across all browsers. The other two technologies, web workers and web storage, being standardized in almost all current browsers, are already exploited successfully in the current web application accessing the CMS experiment condition data.

7.1. Acknowledgments

The authors wish to acknowledge Yan Rymarchik, Mindaugas Polujanskas, and Saulius Bucka from Vilnius University for helping to develop the web application for CMS condition database during the three months ending September 30, 2010. The authors would also like to convey thanks to Salvatore Tupputi, Marco De Mattia, and Jean Fay from RPC, SiStrip, and ECAL, respectively, for developing and implementing the back-end part available through the web application. Their contribution has been very useful to speed up the development of the web application useful for monitoring the calibrations and alignment constants uploaded into the production DB for CMS.

References

[1] HTML5 - W3C Working Draft dev.w3.org/html5/spec/Overview.html
[2] Persistent storage of non-event data in the CMS databases. M. de Gruttola et al. Jan 2010. 20pp. Published in JINST 5:P04003,2010.
[3] PopCon Monitoring: Web Application for Detailed Real-time Database Transaction Monitoring - published in Distributed Multimedia Systems 2009 journal. Ignas Butenas, Salvatore Di Guida, Michele de Gruttola, Vincenzo Innocente, Antonio Pierro.
[4] Data Parallel Programming for the Emerging Web - Richard L. Hudson, Tatiana Shpeisman, Adam Welc, Ali-Reza Adl-Tabatabai - Intel Labs

[5] Kak, Subhash (1986), "Computational aspects of the Aryabhata algorithm", Indian Journal of History of Science 21 (1): 6271.

[6] Safari developer tools. http://developer.apple.com/technologies/safari/developer-tools.html

[7] JavaScript Web Workers Test v1.4.0 - Pedro Verruma - http://pmav.eu/stuff/JavaScript-webworkers/