CMS Conference Report

May 16, 2007

Enhancing the User Interface of the CMS Level 1 Trigger Online Software with AJAX

Ildefons Magrans de Abril
HEPHY, Vienna, Austria
Marc Magrans de Abril¹
CERN, Genève, Switzerland

CMS Collaboration

Abstract

This paper presents a server-side C++ library (AjaXell) for developing graphical user interfaces in Common Gateway Interface compliant servers. AjaXell is based on the Asynchronous JavaScript and XML technology and has been tested in the Apache Tomcat and the Cross Platform Data Acquisition servers.

AjaXell has been envisioned to enhance the browsable user interface of the Trigger Supervisor System developed within the Compact Muon Solenoid (CMS) experiment. A number of feasible technological options to implement the library are discussed. The Trigger Supervisor graphical user interface is presented as an application example. This demonstrates how a web-based desktop-like user interface can be implemented with AjaXell and how this eases the set-up, test, operation and monitoring of the Level 1 Trigger of the CMS experiment.

Presented at *IEEE-NPSS Real Time Conference 2007*, Fermilab, Illinois, USA, April 30, 2007

¹ This work was supported in part by a Marie Curie Early Stage Research Training Fellowship of the European Community's Sixth Framework Programme under contract number (MEST-CT-2005-020216).

Abstract— This paper presents a server-side C++ library (AjaXell) for developing graphical user interfaces in Common Gateway Interface compliant servers. AjaXell is based on the Asynchronous JavaScript and XML technology and has been tested in the Apache Tomcat and the Cross Platform Data Acquisition servers.

AjaXell has been envisioned to enhance the browsable user interface of the Trigger Supervisor System developed within the Compact Muon Solenoid (CMS) experiment. A number of feasible technological options to implement the library are discussed.

The Trigger Supervisor graphical user interface is presented as an application example. This demonstrates how a web-based desktop-like user interface can be implemented with AjaXell and how this eases the set-up, test, operation and monitoring of the Level 1 Trigger of the CMS experiment.

Index Terms—Asynchronous JavaScript and XML, common gateway interface, high energy physics, server side widget library.

I. INTRODUCTION

THE Compact Muon Solenoid (CMS) Online Software Infrastructure (OSWI) is the set of software components responsible for the correct functionality of the CMS experiment. The development of the OSWI has to face many challenges [1]:

- 1) Large size and complexity of the experiment ($O(10^4)$ objects to be controlled).
- 2) Configuration management complexity (many parties involved from around the world with their own preferred software languages, libraries and guidelines).
- 3) Security.
- 4) Long time span of operation (more than 10 years is expected).
- 5) Clients distributed in a large geographical area.

For this purpose, web technologies and in particular web services were chosen as the software system to support interoperability between the different software components through the network. Web services were designed to interconnect highly heterogeneous and distributed systems, and there is a large amount of tools and standards available [2].

Actually the Cross-Platform Data Acquisition middleware (XDAQ) [3] has been chosen by several CMS systems as the web services middleware to deploy the OSWI (references available for the Level 1 Trigger [4], High-Level Trigger [5], Electromagnetic Calorimeter [6], Hadronic Calorimeter [7], and Tracker [8]).

XDAQ version 3 adds a Hypertext Transfer Protocol (HTTP) engine compliant with the Common Gateway Interface (CGI). This new feature turns an executable program into a browsable web application that can visualize its internal data structures [9].

The use of the XDAQ middleware implies that the Graphical User Interface (GUI) implementation should be written in Hypertext Markup Language (HTML) [10] and JavaScript/ECMAScript [11] embed in C++ code. This embedding has two main consequences. First, since the client GUI is a remote web browser without direct access to the hardware, the hardware access code must be decoupled from the GUI code. Usually, this decoupling is facilitated by the use of events between the client and the server. A server-side event-based GUI library will facilitate the decoupling between the controller user interface and the hardware model. Second, there is a cost associated with the learning of HTML and JavaScript/ECMAScript. The developers of the CMS OSWI that use XDAQ must learn two new languages, their syntax, best practices and the testing and debugging methodology using a web browser.

However, these technical problems derived from the embedding of HTML and JavaScript into C++ where not the only items that guide the development of a server-side widget library for C++ CGI applications. Specifically, the library was envisioned as a natural requirement coming from the Trigger Supervisor project (TS) of the Level 1 Trigger (L1 Trigger) [12]. The TS is being used to provide a homogeneous interface to the OSWI of the trigger subsystems in order to set up, test, operate and monitor the trigger components [4].

First, several L1 Trigger subsystems need to integrate board-level control panels in the TS GUI. Initially, these control panels were developed as stand alone tools executed locally or through remote terminals (Java Swing, Tcl/Tk or C++ Qt). The integration of these board-level GUI has to be seen as a step towards the homogenization of the OSWI of the L1 Trigger in order to improve its configuration management (simplifying user's support, maintenance, control of changes, etc.). On the other hand, the integration of the board-level GUI will allow a remote access to a fine grained control of the L1 Trigger through a web browser.

Second, the TS users did requests for new behaviors and views in the TS GUI that usually were seen in their stand alone tools. These requests imply, on one hand, to develop a set of graphical widgets not supported directly by the HTML language, and on the other hand, to allow event-based programming as usually seen in the development tools and libraries of local applications.

After a thorough research, it was realized that no widget library for Linux C++ CGI applications is currently available in the market. For example, the best match between our requirements and the available libraries was the Wt library [13]. However, this library was intended for C++ FastCGI applications under Linux but for CGI (that is the interface available under the XDAQ middleware).

The AjaXell library has been created for this purpose. It thus fulfills the requirements drawn from the TS project (see section 2) and overcomes the consequences of the HTML/JavaScript embedding. The library also has been designed to be used within XDAQ and Apache Tomcat servers in order to maximize the scope of the project and to open a door to future developments. A Source Forge project has been created for this purpose [14].

The remainder of this paper is structured as follows. Section 2 shows the list of requirements and the technological options available to develop the library in the client side. Section 3 presents the library architecture and the main classes. Section 4 presents the usability test of the library in a real scenario developing the GUI of the TS for the Level 1 Trigger. Finally, section 5 summarizes this paper.

II. LIBRARY REQUIREMENTS AND TECHNOLOGICAL OPTIONS

A. Functional and Non Functional Requirements

The functional requirements are listed below:

- 1) Event based programming model. First, to mimic the programming model to which the L1 Trigger typical developer is used (Java Swing and AWT, Tcl/Tk, C++ Qt). Second, to allow users to transparently develop a web-based client/server architecture. Third, simplify the code factorization allowing the library user to implement a per event callback.
- 2) Implement an extended set of graphical widgets (i.e. tab, button, splitter, tree, etc.). The L1 Trigger developers where used to some GUI widgets that can be found in almost any widget library.

The non functional requirements are:

- 3) Modular and extensible. First, the library should allow developers to implement their own widgets in an encapsulated manner (i.e. should allow the creation of C++ graphical widgets classes with their own visualization and behavior). Second, L1 Trigger developers should be able to create custom board-level GUI with a widget-like interface.
- 4) C++ under Linux. The library should be C++ Linux compatible since this is the platform chosen for running the OSWI of the experiment.
- 5) Web browser independent. On the client side the library should work, at least, in the three major web browsers being used in the CMS experiment at CERN: Mozilla/Firefox, Internet Explorer, and Safari.
- 6) CGI server independent. On the server side the library should work within several CGI compliant C++ servers. At least, the library should work within a XDAQ executive and an Apache Tomcat Server.
- 7) Open source based solution. In principle, a non commercial solution should be chosen to avoid vendor lock-in and budgetary restrictions.
- 8) Smooth transition between technologies. The development of the library should not imply a radical change in the set of technologies being used for the developer of the library nor for the users of it. Due to man-power limitations it is preferred a minimum usage of new languages, development tools, etc. Since the previous TS GUI was developed embedding HTML and JavaScript any change on the technology being used should pay back the learning cost.

B. Available Client Side Technologies

Part of the code needs to run on the client-side (web browser). The different technologies that allow for this are presented below [15].

1) *Java Applets*: An Applet is a program written in the Java language that can be embedded on a web page. These code snippets require the proprietary Java plug-in from Sun to run [16]. This proprietary plug-in is freely distributed as the tools to develop Java applications. There are two main disadvantages on using Java Applets in the context of the L1 Trigger OSWI. First, it is not possible to load a complex Java Applet from the XDAQ server (i.e. complex in the sense that the .jar file is bigger than 1Mb). Second, it is required to learn and maintain an additional programming language within the TS project. This implies an increase on the cost and complexity of the configuration management of the OSWI.

2) *XUL and XAML markup languages*: Mozilla Foundation's XUL (XML-based user interface language) [17] and the Microsoft Internet Explorer's counterpart XAML (XML-based user interface language) [18] are two XML conformant languages used to define objects and their properties, relationships and interactions in the web browser. Although these languages offer a high-performance, fast, and robust behavior, these have the main disadvantage of being

3) *Adobe Flash*: Flash is a proprietary technology of Adobe Systems Inc. [19], normally it refers to both the Adobe Flash Player and to a multimedia authoring program used to create content for the Adobe Engagement Platform (such as web applications, games and movies). The Flash Player is a client application available in most dominant web browsers as a free

Fig. 2. Sequence diagram of the basic interaction between the web browser (client) and the server using the AjaXell library.

Fig. 1. Graphical representation of the basic interaction between the web browser and the server using the AjaXell library.

plug-in, supports for vector and raster graphics, a scripting language called ActionScript, and bi-directional streaming of audio and video. The main advantages of Flash are the web browser and platform compatibility (almost all the important browsers have the corresponding plug-in to run the .swf flash files), speed, flexibility, and increasingly powerful development tools. Although there is a complete set of tools and libraries for Flash and related technologies (some of them open source), the developers will need to face two main problems: vendor lock-in and the learning cost of a new set of skills (development tools, ActionScript language, etc.).

4) *Asynchronous JavaScript and XML (AJAX)*: The term first appeared in a paper from 2005 [20] and the first attempts to use in the HEP community comes from 2006 [21]-[22]. AJAX is a set of standard technologies combined in a given way to create interactive and desktop-like applications:

- 1) HTML and Cascading Style Sheets (CSS) [23].
- 2) The Document Object Model (DOM) accessed with a client-side scripting language like ECMAScript. The implementations of the ECMAScript access to the DOM object allows to dynamically display and interact with the information presented.
- 3) The XMLHttpRequest object used to exchange data asynchronously with the web server [24].
- 4) XML is sometimes used as the format for transferring data between the server and client while executing the XMLHttpRequest. Although any format will work, including non formatted HTML/JavaScript, plain text, JSON (JavaScript Object Notation), etc.

The AJAX technology has been chosen as the most suitable because:

- 1) Efficient use of bandwidth. AJAX allows the interchange of small amounts of data between the client (web browser) and the server that corresponds to the incremental change in the web page. Therefore, it is not necessary to send again all the web page every time a request is done in the server.
- 2) Improved responsiveness. The web browser is less time blocked each time an action is executed in the web browser because the HTTP request is asynchronous in the AJAX case.
- 3) Allows an event-based model and improves the responsiveness of the web page. The XMLHttpRequest facilitates an event based model in the server. The events of the DOM tree can be connected to C++ code in the server. After an event is fired in the browser an XMLHttpRequest object is created and an asynchronous call to the server is done. The server response refreshes a small portion of the web page dynamically (see section 3.1 for an explanation of the basic mechanism).
- 4) It is possible to use available client-side libraries. It is possible to choose among a large set of open source and commercial solutions that implement the AJAX behavior in the client-side [25]. Actually, the AjaXell library is using an open-source JavaScript library named Dojo [26].
- 5) Smooth Transition between technologies. The developers of the AjaXell library will use almost the same technologies as those in the first GUI developed for the TS (XHTML, CSS, and JavaScript). The only additions will be the manipulation of

the DOM tree dynamically using JavaScript and the use of XMLHttpRequest object to communicate with the server. Once developed, the users of the AjaXell library will not need to test or debug client-side code. The users of the library will write server-side C++ code, and transparently, the client-side AJAX code will be generated.

- 6) Standard based solution. AJAX is based on open standards of the World Wide Web Consortium (W3C) [27].

III. ARCHITECTURE

A. AjaXell Library Basic Intercation

AjaXell is a library based on AJAX. Therefore the actions performed in the web browser will execute an asynchronous call to the server that will refresh a small portion of the web page.

This is done through the following steps (see also the basic interaction as a graphical representation in Figure 1 or as a sequence diagram in Figure 2).

- 1) A DOM event is triggered in the browser after some action of the user (e.g. someone clicks a button).
- 2) The DOM event is redirected to a JavaScript handler that was previously generated by AjaXell and loaded in the browser.
- 3) The JavaScript handler creates an XMLHttpRequest object and performs an asynchronous call to the server. An identifier of the widget, the event type, and the value in the corresponding widget (if exist) are sent with the request.
- 4) On the server, the widget identifier and the event type are read and used to redirect the request to the appropriate event handler within a method of a C++ class.

Fig. 3. UML class diagram of the EventHandler class. The HTTP request information is passed to the callback through a `cgicc::Cgicc` object and the HTTP response is written in a `std::ostream`.

Fig. 4. UML class diagram of the AutoHandledWidget and related classes. This class inherits from the Widget and Container classes. It has an EventHandler.

- 5) The C++ event handler creates the response using the AjaXell library and sends back to the browser HTML and JavaScript code that corresponds to new widgets and events.
- 6) The reply arrives to a JavaScript handler that dynamically modifies a portion of the DOM tree that represents the web page.

B. Library Architecture

The AjaXell library is structured around four classes (see Fig. 4): Widget, EventHandler, AutoHandledWidget and Page. The first two classes encapsulate the basic look and feel functionalities. The other two classes are a combination of the previous two and allow the creation of widgets (and web pages) with the server-side event handler code encapsulated within.

1) *Widget Class*: The Widget class is the root element of the AjaXell library. A widget instance can be visualized in a web browser, so the widgets can be encoded in a HTML and JavaScript representation through the Widget::html method. All the Widgets descendant classes must override the Widget::html method in order to encode the corresponding look and feel of it.

Multiple inheritance is used in the library. Although there are widgets that are direct descendants from the Widget class (e.g. Image, Graph, etc.), some of them can, at the same time, be descendants of the Eventable or Container classes (see Table I). An Eventable widget can have one or more handlers that reply to a DOM event (e.g. Button, TreeNode, Tab, InputText, etc.).

2) *EventHandler Class*. The EventHandler class dispatch the appropriate callback for a given widget identifier and event type. The callback can be a method of any C++ class with a specified signature. A Cgicc object [28] is passed to the callback to parse the HTTP request. The HTTP response is written by the callback in a C++ standard output stream.

3) *AutoHandled Widget Class*. The AutoHandled Widget class allows code modularization in the customization process of the

TABLE I
WIDGET DESCENDANT CLASSES

Parent Classes	AjaXell Classes
Widget	Graph, BarGraph, ScatterGraph, Frame, PlainHtml, Table, SortableTable
Widget, Eventable	Button, InputText, InputPassword, InputFile, Slider, FishEyeMenuItem, RadioButton, CheckBox
Widget, Container	TabContainer, ResultBox, Dialog, Tree, FishEyeMenu, RadioGroup, Splitter, LayoutContainer, LayoutElement
Widget, Eventable and Container	Form, TreeNode, Tab

GUI. It is the base class for any custom made widget. It allows developers to create a class that encapsulates the look and feel of a new widget. This means that any descendant of AutoHandledWidget is able to contain not only widgets themselves (i.e. HTML and JavaScript code embedded in C++) but also server-side C++ handlers that are connected to a DOM event of the browser.

An AutoHandledWidget is also a Container, so it is possible to embed other generic Widgets and AutoHandledWidget inside one of them in a recursive way.

4) *Page Class*: The Page class is an AutoHandledWidget descendant that contains all the Widgets of the web page. The html method of this class visualizes the whole page instead an incremental portion of it. This class can contain event handlers that corresponds to the various events within.

Fig. 6. Screenshot of the new TS GUI. The GUI is accessible from a web browser and integrates the many services of the TS in a desktop-like fashion.

IV. APPLICATION SCENARIO: TS GUI

A usability test has also been performed in a real scenario. The GUI of the TS project has been fully developed using the AjaXell library and currently is being used by all the subsystems of the L1 Trigger. The aim of that GUI is to integrate the many services offered by the TS in a single browsable application. The services that has been integrated in the TS GUI include the set-up, test, operation and monitoring of the Level 1 Trigger of the CMS experiment, the population of the configuration Data Base for any of the L1 Trigger subsystems (see [29]), access to support, documentation, and finally, access to the log record for audit trials and postmortem analysis.

In Fig. 5 and 6 it is possible to appreciate the differences between the old GUI of the TS and the new GUI based on AjaXell. The main differences are:

- 1) The new GUI works as a single-page interface like a desktop application.
- 2) The number of actions and graphical elements in the page has experienced a ten fold increase (from 16 actions and $O(10)$ elements per page in the old GUI to a minimum of hundred events and elements in the new one).
- 3) The event handlers only modify the desired part of the web page increasing the responsiveness and efficient use of the bandwidth.
- 4) The use of tabs, trees and other graphical elements has increased the information available in the same web page. This has improved the user experience.

V. CONCLUSION

AjaXell is a server-side C++ Linux library to create CGI applications running in XDAQ and Tomcat Apache servers. After the discussion depicted in section II.B the AJAX technology was chosen as the most appropriate. AJAX is based on W3C standards, improves the responsiveness of the web applications, has a large open source community working and supporting it, and has proven successfully in applications like Gmail, Flickr, or 24SevenOffice.

The library has been used to integrate all the services of the Trigger Supervisor in a single desktop-like web application.

It is possible that the same requirements faced while developing the library will be common to all the low level CMS OSWI that uses the XDAQ middleware and it is also possible that the use of AJAX widespread over the HEP community as shown in first attempts [21]-[22]. Therefore, in order to achieve the maximum scope the library has been decoupled from the TS code and has been designed to be compatible with Tomcat Apache CGI Applications.

ACKNOWLEDGMENT

We would like to acknowledge Alexander Winkler for his initial work on the TS GUI.

REFERENCES

- [1] V. Brigljevic et al, "Run Control and Monitor System for the CMS Experiment", *Computing in High Energy and Nuclear Physics*, 2003.
- [2] CMS Collaboration, *The TriDAS Project – Data Acquisition and High-Level Trigger Technical Design Report*, CERN/LHC, 2002.
- [3] V. Brigljevic et al, "Using XDAQ in Application Scenarios of the CMS Experiment", *Conference for Computing in High-Energy and Nuclear Physics*, 2003.
- [4] I. Magrans, M. Magrans, "The CMS Trigger Supervisor Project", *IEEE Nuclear Science Symposium Conference*, 2005.
- [5] V. Brigljevic et al, "The CMS High Level Trigger", *IEEE Nuclear Science Symposium Conference*, 2005.
- [6] The ECAL DAQ Software Wiki, <http://twiki.cern.ch/twiki/bin/view/CMS/EcalDAQSoftware>.
- [7] J. Mans and W. Fisher, "Data Acquisition Software for CMS HCAL Testbeams", *Computing in High Energy and Nuclear Physics Conference*, 2003.
- [8] G. Iles et al, "A Testing Device for the CMS Silicon Tracker Front End Driver Cards", *LHC Electronics Workshop*, 2003.
- [9] R. Arcidiacono et al, "Hyperdaq – Where Data Acquisition Meets the Web", *Conference on Accelerator and Large Experiment Physics Control Systems*, 2005.
- [10] Document Object Model Level 2 HTML Specification, World Wide Web Consortium, 2003.
- [11] ECMAScript Language Specification, Standard ECMA-262, 1999.
- [12] I. Magrans, C. Wulz and J. Varela, "Concept of the CMS Trigger Supervisor", *Compact Muon Solenoid Note*, 2005/011.
- [13] Wt: C++ Web Toolkit Project v. 1.99, <http://sourceforge.net/projects/witty/>.
- [14] AjaXell Source Forge project, <http://sourceforge.net/projects/AjaXell/>.
- [15] A. White, "Weighing the Alternatives", AJAX Info, <http://www.AJAXinfo.com/default~viewart~8.htm>, 2005.
- [16] K. Hopson and S. Ingram, *Developing Professional Java Applets*, Sams Publishing, 1996.
- [17] D. Hyatt, *XML User Interface Language Specification*, <http://www.mozilla.org/projects/xul/>.
- [18] Extensible Application Markup Language Web Page, <http://msdn2.microsoft.com/en-us/library/ms747122.aspx>.
- [19] Adobe Flash Web Page, <http://www.adobe.com/products/flash/>.
- [20] J. J. Garret, "AJAX: a New Approach to Web Applications", <http://www.adaptivepath.com/publications/essays/archives/000385.php>, February 2005.
- [21] Aleverson et al, "Interactive Web-Based Analysis Clients Using AJAX: With Examples for CMS, ROOT and GEANT4", *Computing in High Energy and Nuclear Physics*, 2006.
- [22] Subir Sarkar, "A Web based Histogram Viewer Application built using the AJAX Technology", *ROOT User Workshop*, 2007.
- [23] B. Bos et al, *Cascading Style Sheets W3C Level 2 Specification*, <http://www.w3.org/TR/CSS21/>, 2006.
- [24] A. Van Kesteren, *The XMLHttpRequest Object*, <http://www.w3.org/TR/XMLHttpRequest>, 2007.
- [25] Survey of AJAX/Javascript Libraries, <http://wiki.osafoundation.org/bin/view/Projects/AjaxLibraries>.
- [26] Dojo, the Javascript Toolkit, <http://www.dojotoolkit.org>.
- [27] World Wide Web Consortium, <http://www.w3.org/>.
- [28] GNU CGICC library, <http://www.gnu.org/software/cgicc/cgicc.html>.
- [29] I. Magrans, L. Van Nierop, S. Iqbal, "A Database Interface Based on Java Server Pages Technology for the CMS Experiment", *Compact Muon Solenoid Internal Note*, 2005/036.