Available on CMS information server **CMS NOTE -2010/005**

The Compact Muon Solenoid Experiment

CMS Note

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

23 July 2007 (v4, 19 January 2010)

Homogeneous User Interface Infrastructure for Expert Control of the Level-1 Trigger

M. Magrans de Abril, I. Magrans de Abril, , C. Hartl, A.-J. Winkler, M. Jeitler, I. Mikulec, C.-E. Wulz

Abstract

This note presents a building block of the Trigger Supervisor, the control system of the CMS Level-1 Trigger, which provides expert user interfaces for hardware control and monitoring. This software infrastructure allows the development of web-based plug-ins for the Trigger Supervisor that extends its generic user interface. The integration of expert tools using the new facility will be a step towards the homogenization of the Level-1 Trigger software. This document also specifies the aim, requirements, and a detailed description of the control panel architecture and performance. Finally, the Global Trigger control panel is presented as a use case of the infrastructure's viability.

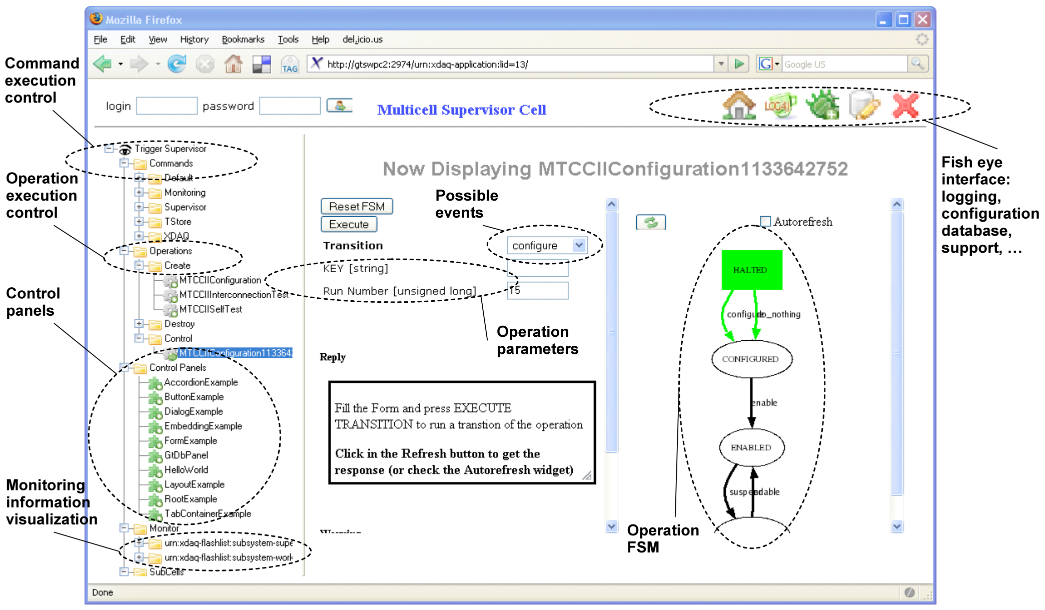
1 Introduction

The Level-1 Trigger (L1) is a custom hardware system that along with the High Level Trigger (HLT) reduces the data rate generated by the Compact Muon Solenoid (CMS) experiment [1]-[3]. The Trigger Supervisor (TS) is a distributed software system based on the XDAQ middleware [4]. It controls the operation of the Level 1 Trigger of the CMS experiment. The TS is a distributed control system featuring a hierarchical topology of nodes or cells. Both the top and subsystem cells provide homogenous remote machine access and a generic graphical user interface (GUI). Currently, TS-enabled nodes control all the L1 Trigger and part of the Timing, Trigger and Control System (TTC) system (i.e. 9 hardware subsystems, deployed in 64 crates, controlled by 62 TS nodes, and more than 1000 boards). Around 20 persons have been developing this control system for the last 4 years.

When a XDAQ executive is launched, the TS Cell and a number of pluggable components are dynamically loaded in order to provide basic functionalities (i.e. XDAQ applications). One of the main XDAQ applications is Hyperdaq. It facilitates a web interface that turns an executive into a browsable web application, able to provide access to the internal data structure of any XDAQ application loaded in the same executive [5]. Any XDAQ application can customize its own web interface by developing Hypertext Markup Language (HTML) and JavaScript code embedded in C++. Mixing three different languages in the same program has a cost associated with the learning curve because developers must learn two new languages, their syntax, best practices and the testing and debugging methodology using a web browser.

AjaXell [6] is a C++ software library intended to smooth this learning curve. This library provides a set of graphical objects named widgets like sliding windows, drop down lists, tabs, buttons, dialog boxes and so on. These widgets ease the development of web interfaces with a look-and-feel and responsiveness similar to the stand-alone tools executed locally or through remote terminals.

Figure 1 shows the web interface of a cell implemented with the Ajaxell library. This is an out-of-the-box solution that does not require any additional development by the subsystems and assures the cross-browser compatibility (currently it supports Firefox 1.5 to 3.x, Internet Explorer 7.x to 8, and Chrome 1.x). It provides several controls: to execute cell commands; to initialize, operate, and kill cell operations; to visualize and retrieve monitoring information; to access the logging record for audit trials and postmortem analysis; to populate the L1 trigger configuration database; to request support; and to download documentation.



Fill the Form and press EXECUTE TRANSITION to run a transtion of the operation

Click in the Refresh button to get the response (or check the Autorefresh widget)

Figure 1: Generic GUI of the TS. The central panel shows the generic human interface of an operation.

However, the default TS GUI is not sufficient to cope with the high complexity and granularity of the trigger system. The TS control panel was developed to fill this gap. A control panel is a subsystem specific graphical setup, normally intended for expert operations of the subsystem hardware. The control panel infrastructure allows to develop expert tools with the TS framework. This new infrastructure allows the migration of existing standalone graphical tools (e.g. C++ Qt, Java Swing, Microsoft Foundation Class Library, Tcl/Tk, etc.) to web-based control panels and, therefore, contributes to the homogenization of the L1 software, which provides the following benefits:

i. Shortened learning curve of the operators. They always find the same kind of graphical look and feel regardless of the L1 subsystem being operated.

ii. Simplified maintenance of the overall L1 software infrastructure. The standalone tools, which were previously implemented in different languages and widget frameworks, are gradually substituted by a unique software approach (i.e. Java Swing, C++ Qt and Tcl/Tk tools will be substituted by C++ TS control panels).

iii. Sharing of software and experience between subsystems.

The present document is divided in five sections. Section 1 defines the context of the project. Section 2 describes the functional and non-functional requirements of the control panel infrastructure. Section 3 outlines the implementation of the infrastructure, explains a prototypical "Hello World!" example, and the performance of the infrastructure implementation. Section 4 shortly describes the Global Trigger (GT) control panel case as an example. Finally, section 5 summarizes the achievements and consequences.

2 Requirements

The requirements discussed below emerged from the experience with existing GUI libraries, the current use cases of already existing standalone tools used to operate various subsystems, and the interaction with the users.

2.1 Functional Requirements

2.1.1 Integration with TS and Subsystem Layers

The control panels should allow the integration of the functionalities provided by the TS Framework layer and the subsystem layer. This should include the control and monitoring of hardware through TS facilities (i.e. commands and operations), online database access, and communication with other TS and generic XDAQ nodes.

2.1.2 Event-Driven Model

Currently, the event-driven model is the preferred one amongst the GUI libraries (e.g. C++ Qt, Java Swing, Microsoft Foundation Class Library, Tcl/Tk, etc.). In this model the flow of the program is not specified a priori but determined by the user actions. The main advantages of event driven programming over polling become visible when large amounts of data needs to be processed incrementally and when there are multiple event sources. This is exactly the case for GUI libraries.

2.1.3 Desktop-Like Look and Feel

In software design, the term "look and feel" is used with respect to a graphical user interface and comprises aspects of its design, including elements such as colors, shapes, layout, and typefaces (the "look"), as well as the behavior of dynamic elements such as buttons, boxes, and menus (the "feel"). It is desirable that the control panel infrastructure gives a user experience similar to the one of the standalone tools running locally.

In practice, this means that common widget classes should be available to customize the control panel (buttons, menus, expandable trees, layouts, accordions, etc.), in order to allow for a GUI with features similar to those of the standalone tools that the experts are used to work with.

2.2 Non-functional Requirements

2.2.1 Acceptable Response Time

There are two time intervals that allow measuring the performance from the user point of view: the loading time and the event response time. The first refers to the interval between the user request of a control panel (e.g. clicking an icon) and the display of the control panel in the browser. The second refers to the interval between an action performed by the user on the control panel and the display of the result. Both measures of performance are subjective and literature on what are an acceptable numbers for them is not conclusive [7]. With that in mind we quantify the performance requirements as follows:

- *The loading time should be less than 10 seconds for ~1000 visual elements (e.g. html input buttons)*:

This is about the limit for keeping the user's attention focused on the dialogue with the computer.

- *The event response time should be less than 1 second for ~1000 visual elements*: This is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. On the other hand, a response time below 0.1 seconds is about the limit for making the user feel that the system is reacting instantaneously.

2.2.2 Enforce Encapsulation, Modularity, Compactness, and Layering

It is always desirable to enforce encapsulated, modular, compact and layered implementations of software. These characteristics improve the quality of the source code (and thus account for maintainability, experience sharing, reduced number of errors, etc.). It is possible to enforce these characteristics by constraining the design decisions in the following ways:

- The control panel graphical design and event handler information should be stored in the same class (enforce encapsulation).
- It should be possible to decompose a complex control panel into several simpler ones. Each of them should store their own design and event handlers (enforce modularity).
- The number of lines of source code needed to create a control panel should be minimal. A simple example should not include redundant information (enforce compactness).
- The event handlers should have access to the subsystem, TS and XDAQ layers (enforce layering).

3 Architecture, Simple Example, and Performance

3.1 Architecture

The TS integration methodology is a process with two steps. The first step creates a set of high level services that control and monitor the hardware of one of the L1 subsystems. These services provide an abstraction mechanism of the OS and hardware drivers. The second step customizes the generic GUI, which allows for fine-grain control by experts and operators. The development of subsystem specific control panels is part of the second step. In this context, the control panel developer needs to be familiar with the AjaXell library, the TS framework and the available subsystem services. Previous knowledge of the operating system particularities, the hardware drivers, or the XDAQ middleware is not required.

Figure 2 shows the class structure resulting from the customization process of a simple subsystem. In this case, the subsystem node of the TS System has created two high level services: a configuration operation (i.e. a descendant of `ts::framework::CellOperation` named `subsystem::MTCCIIConfiguration`) and a command to read a concrete VME register (i.e. a descendant of `ts::framework::CellCommand` named `subsystem::readsTTS`). Initially, the TS Framework provides a generic HTTP/CGI interface to these services (a SOAP interface is also provided). A control panel can be created to develop a custom look and feel of the high level services, which is adequate for the given subsystem.

The control panel will access the hardware control and monitoring services through the corresponding command or operation factory located in the `subsystem::CellContext` and will create the web design using the AjaXell library.

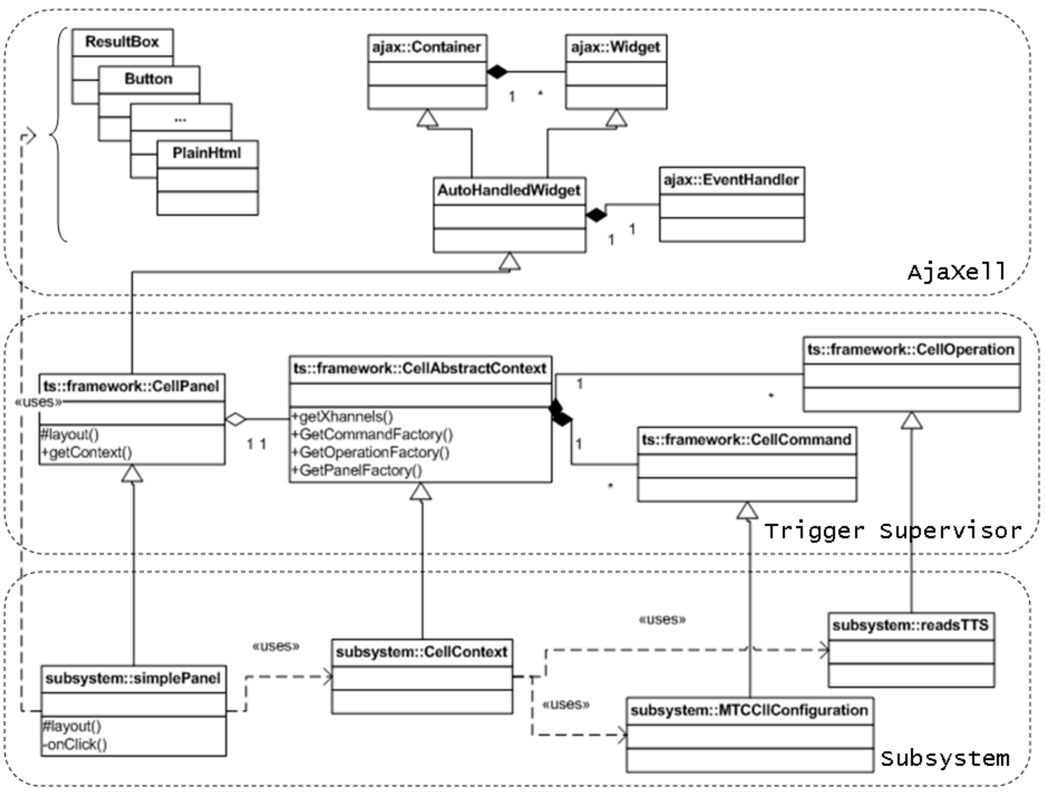


Figure 2*:* UML Class diagram of the relationships between the AjaXell library, TS framework and a prototypical subsystem (see section 3.2 for the source code of the `subsystem::simplePanel` class).

3.2 Simple Example

This section presents a simple control panel example in order to demonstrate that the non-functional requirements of encapsulation, modularity, compactness and layering have been fulfilled.

Figure 3 shows a complete example for a control panel. When the button of the control panel is clicked a "Hello World!" phrase appears inside the bottom result box (Figure 4). This example has been created with two classes: `SimplePanel` and `HelloWorld`. The same functionality can be created with only one class, however, in this way it will be obvious that the infrastructure allows an encapsulated, modular, compact and layered design:

- *Encapsulation* and *modularity*. The `SimplePanel` class has two methods and one attribute (apart from the constructor). The `SimplePanel::layout` method is used to build the initial look and feel of the example (i.e. what is shown when the control panel is loaded or when the browser is refreshed). The `SimplePanel::onClick` method is the server-side handler for the OnClick DOM (Document Object Model) event of the HTML button created by the first method. The server-side handler will send back the "`<p>Hello World!</p>`" phrase using the `HelloWorld` "sub-panel" class. Finally, the `result_` attribute is the server side representation of the result box (i.e., a `<div>` element with an `id` attribute) where the phrase is displayed. Therefore, the visual design and the server-side event handlers are encapsulated in a single class (i.e. the look and feel is encapsulated). It is modular as we are able to decompose several parts of the web page into different classes.
- *Compactness*. The design of the control panel infrastructure also allows a compact program. As can be seen in Figure 3, 40 lines suffice to create a complete example (and just 10 lines for a "Hello World!" control panel).
- *Layering*. The example shown in Figure 3 does not have XDAQ or OS specific source code, the program does not need to interact with the XDAQ middleware. Therefore, the developer just needs to know the subsystem specific program, the TS layer and the usage of the AjaXell library. In this example, the subsystem software layer is not exercised. A simple call to `getContext()-`

>getCrate()->getStatus() could exemplify the usage (more details about the interaction with the subsystem layer are given in section 4).

Finally, in order to complete the description of the example, Figure 5 shows the interaction sequence between different software layers after clicking on the “click me!” button:

- The onClick DOM event arrives at the AjaXell JavaScript handler.
- The handler makes an XMLHttpRequest to the XDAQ server through its HTTP/CGI interface.
- The server redirects the call to a single entry point in `CellAbstract::Default` method.
- The `CellAbstract` method redirects the request to the corresponding C++ handler using the id and event type passed through the XMLHttpRequest.
- The request arrives to the `SimplePanel::onClick` and the method returns an output stream with the phrase “`<p>Hello World!</p>`”.
- The reply arrives asynchronously to an AjaXell Javascript handler that refreshes the content of the corresponding node of the DOM tree.

```
class SimplePanel: public CellPanel {
  public:

    SimplePanel(CellAbstractContext* context, log4cplus::Logger& logger)
      :CellPanel(context,logger), logger_(Logger::getInstance(logger.getName() +".SimplePanel")) {};

  private:

    ajax::ResultBox* result_;

    void layout(cgicc::Cgicc& cgi) {
      remove();
      ajax::PlainHtml* title = new ajax::PlainHtml();
      title->getStream() << "<h2 >Now Displaying subsystem::SimplePanel</h4>" << std::endl;
      add(title);

      result_ = new ajax::ResultBox();
      result_->set("style","margin:20px; padding:20px; border:2px solid; ");

      ajax::Button* button = new ajax::Button();
      button->setCaption("click me!");
      setEvent(button,ajax::Eventable::OnClick,result_,this,&SimplePanel::onClick);
      add(button);

      add(result_);
    };

    void onClick(cgicc::Cgicc& cgi,std::ostream& out) {
      result_->remove();
      HelloWorld* hello = new HelloWorld();
      result_->add(hello);
      result_->innerHtml(cgi,out);
    };
}

class HelloWorld: public CellPanel {

  public:

    HelloWorld(CellAbstractContext* context, log4cplus::Logger& logger)
      :CellPanel(context,logger), logger_(Logger::getInstance(logger.getName() +".HelloWorld")) {};

  private:

    void layout(cgicc::Cgicc& cgi) {
      ajax::PlainHtml* plain = new ajax::PlainHtml();
      plain->getStream() << "<p>Hello World!</p>";
      add(plain);
    };
};
```

Figure 3*:* Complete example for the SimplePanel control panel. The `SimplePanel::layout` method contains the initial look and feel. The `SimplePanel::onClick` method contains the handler for the onClick event of the browser DOM tree. The `HelloWorld` control panel is embedded in the `subsystem::SimplePanel`.

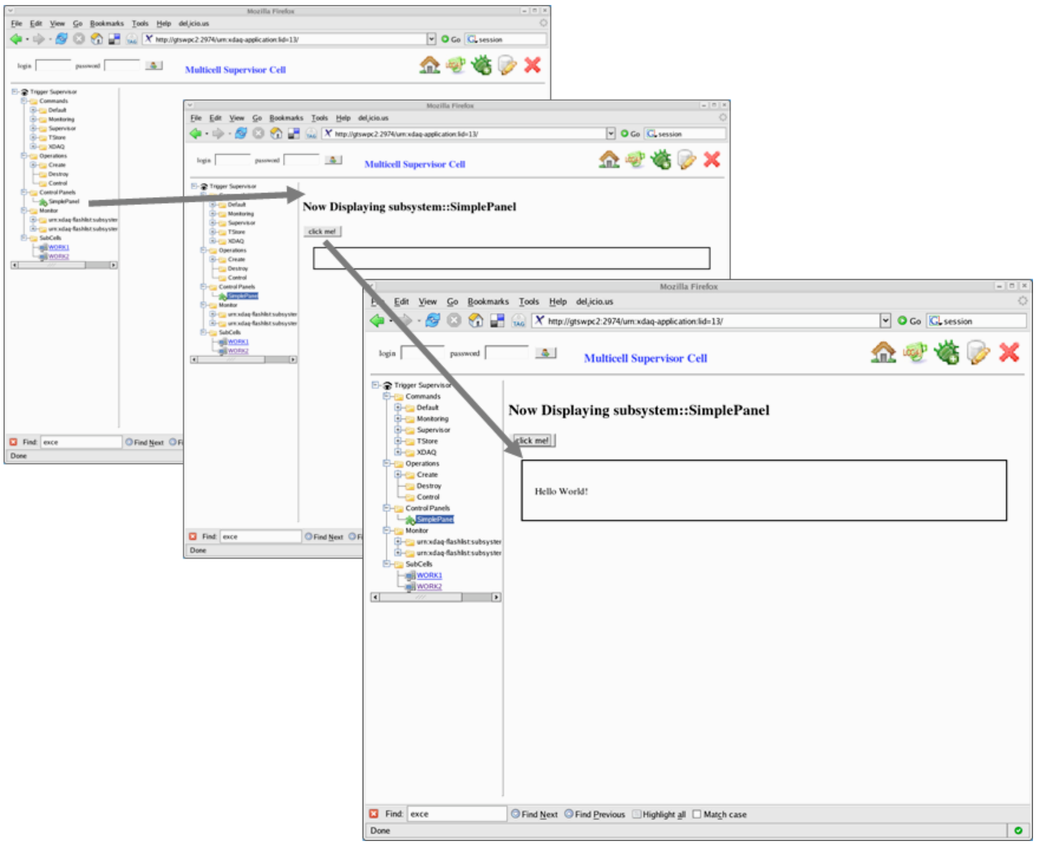

Multicell Supervisor Cell
Now Displaying subsystem::SimplePanel
click me!
Hello World!

Figure 4: Sequence of screenshots for the `subsystem::SimplePanel` example. First, an instance of the control panel is created and visualized after clicking the explorer tree on the left side. Second, the user clicks on the "click me!" button. Finally, the box content is changed to "`<p>Hello World!</p>`".

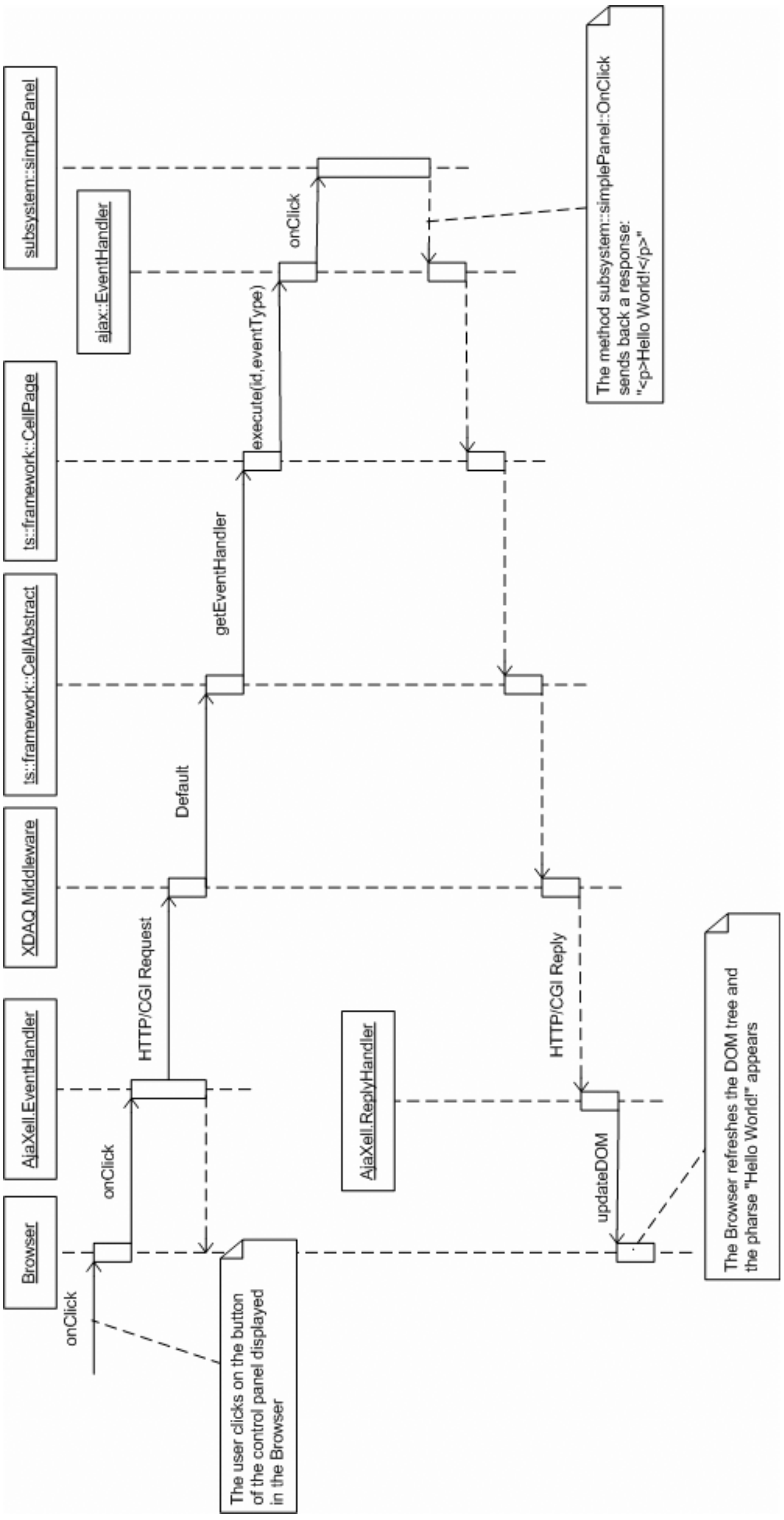

Browser
AjaXell.EventHandler
XDAQ Middleware
ts::framework::CellAbstract
ts::framework::CellPage
ajax::EventHandler
subsystem::simplePanel
AjaXell.ReplyHandler
onClick
onClick
HTTP/CGI Request
Default
getEventHandler
execute(id,eventType)
onClick
HTTP/CGI Reply
updateDOM
The user clicks on the button of the control panel displayed in the Browser
The method subsystem::simplePanel::OnClick sends back a response: "<p>Hello World!</p>"
The Browser refreshes the DOM tree and the pharse "Hello World!" appears

Figure 5: UML sequence diagram of the interaction of software components after a mouse click on the simple example control panel button.

3.3 Performance

Performance measurements have been carried out to test the responsiveness of the TS control panels. The results have been compared with the requirements stated in section 2.2.1.

To measure the responsiveness we have used two different computers running in the CMS private network (Ethernet at 1 Gbps). The control panel server runs with the TS Framework version 1.6 on CERN Scientific Linux 4, with an Intel Xeon at 2GHz, and 4GB RAM. The control panel client runs on a Mozilla Firefox 2.0.0.14 launched also on Scientific Linux 4, with an Intel Xeon 2.66Ghz, and 16GB RAM.

A test control panel has been implemented to test the responsiveness of the software. This control panel has a single callback associated to an onchange DOM event of an HTML text input. When a number is written in the HTML text input (of the client) the control panel callback is executed (on the server). The callback generates as many AjaXell buttons (and its corresponding events) as the number written down in the text input. Then the AjaXell library generates the corresponding HTML/JavaScript code that is sent back to the web browser. Finally, the web browser parses, evaluates and renders the HTML/JavaScript code of the HTTP reply to display the buttons.

The AjaXell buttons have been used as approximation to a GUI average component. A button complexity lies between the HTML phrase, heading or link, and the one of a Tab or Accordion element. A button is also the prototypical component of a dynamic web page.

Figure 6 shows the browser response time as a function of the number of buttons sent or the amount of data transferred. It can be observed that the requirements stated in section 2.2.1 are fulfilled when:

- If the number of GUI components within a control panel is less than 8000, then the loading time will be less than 10 seconds. In that case the user will not lose focus.
- If the number of GUI components to be refreshed by an event is less than 1000, then the event response time will be less than 1 second. In that case the user will not interrupt the flow of thought.
- If the number of GUI components to be refreshed by an event is less than 64, then the perceived refresh time will be instantaneous.

The amount of GUI components allowed fulfills the current CMS operations use cases (see section 4 for GT case).

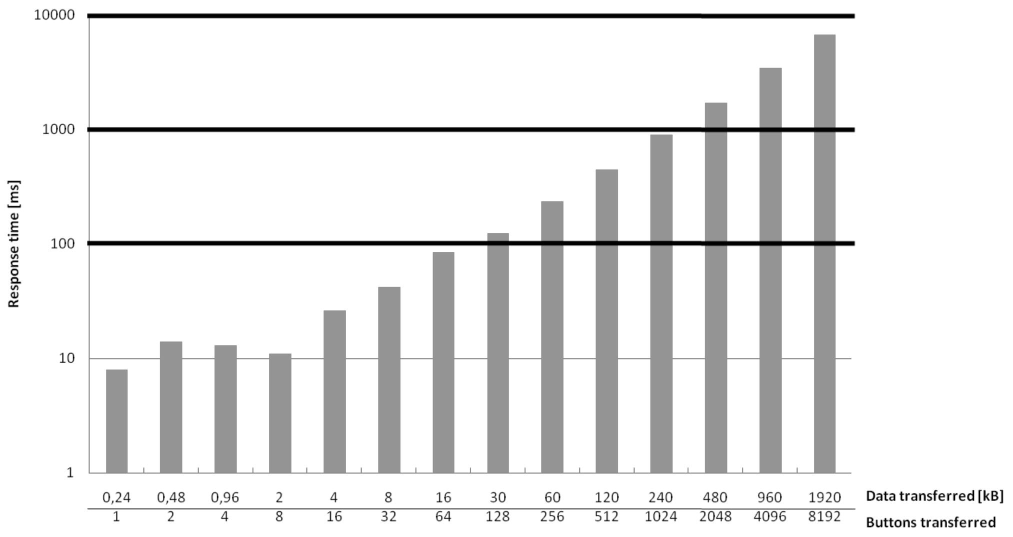


Figure 6: Event response time as a function of the amount of data transferred or the number of buttons generated in the server and then parsed, evaluated and rendered in the web browser.

It is also possible to decompose the response time in its various components as a function of the event size. Figure 7 shows that the browser parsing, evaluation and rendering of the HTML/JavaScript code is the major cause of an increase in the response time for medium and large events. The bars show the contribution on the response time of the different components (from top to bottom: web browser, network, TS, XDAQ).

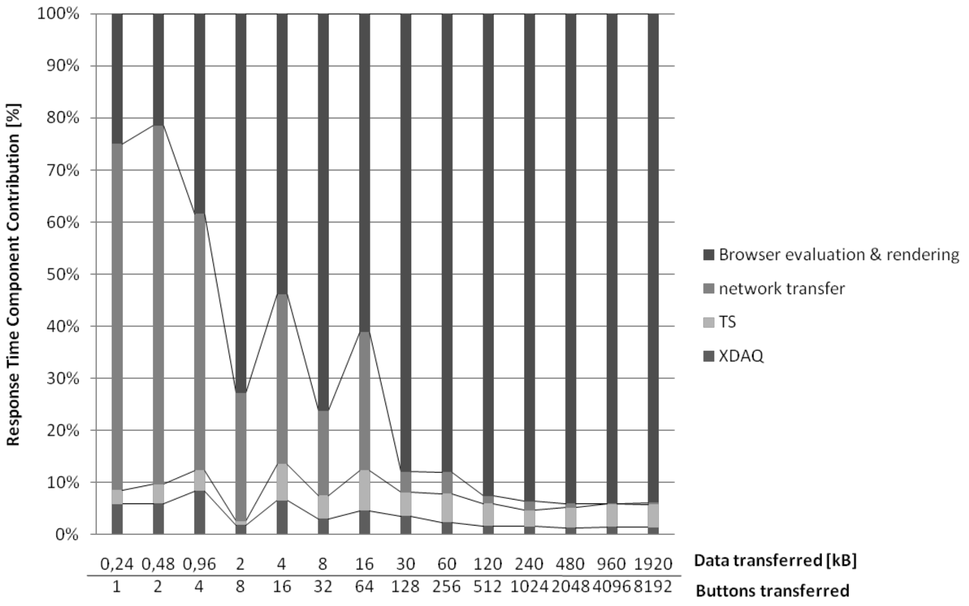


Figure 7: Response time decomposition (in percentage) as a function of the amount of data transferred (or the number of AjaXell buttons transferred). Four components are shown: XDAQ middleware, TS Framework and test control panel, network I/O, and browser parsing, evaluating and rendering of the HTML/JavaScript code.

4 Case Study: The Global Trigger Control Panel

4.1 Introduction

The Global Trigger (GT) [9] is the last stage in the CMS Level-1 trigger chain. Its main task is to apply a set of algorithms (Level-1 Trigger Menu) to input trigger objects and produce "Level-1 Accept" (L1A) decisions according to a pre-defined trigger menu. The input to the GT consists of Global Calorimeter Trigger (GCT) and Global Muon Trigger (GMT) objects, as well as additional technical trigger inputs.

The Trigger Control System (TCS) is part of the Global Trigger hardware. It distributes fast control signals (such as the LHC-orbit marker used for synchronization purposes) and L1As to 32 detector partitions, and it takes into account the Trigger Throttling System (TTS) status of these partitions. That is, it reacts to states such as "Busy", "Out-of-Sync", etc. as appropriate. By design, the TCS can be partitioned into up to eight partition groups that can be independently operated.

When the GT triggers an event by sending a Level-1 Accept (L1A) signal, this causes not only the connected subdetectors to be read out, but also the GT itself. Data from all GT boards, including the GMT, are collected in a single readout module, the GTFE (Global Trigger Front End), which forwards the event data to the Data Acquisition System (DAQ).

The GT online software consists of the GT Cell, a TS/XDAQ web application [10], and auxiliary services. The GT Cell contains several control panel plug-ins; we will refer to them in the following by the name "GT Control Panel" (GTCP). These panels provide interaction with hardware and databases. They are customizations of the TS generic GUI, as described earlier.

The following sections outline the requirements on the GT Control Panel and describe its current implementation. Results of response time performance measurements are presented.

4.2 Requirements

4.2.1 Functional Requirements

Monitoring and control of status information

The status of connected detector partitions and the GT boards themselves must be monitored. Graphical information about the input, output and Finite State Machine state of the eight TCS partition controllers is needed as well.

For the GT boards and any connected detector or trigger subsystem, the input status can also be ignored in the GT, i.e. forced to “Ready”. Therefore, it should be displayed for all relevant entities, whether the status is real or enforced. It must be possible to turn enforcement on and off with the GUI.

Monitoring and control of detector partitioning

The GUI must inform about which detector partitions are connected to which TCS partition controller (PTC, corresponding to a DAQ partition). The GT itself can either contribute to the data stream for all eight partitions, or it can send data only for a given partition under test. This needs to be displayed as well.

The time slots, which are assigned to the eight partition controllers, must be displayed and should be modifiable in the GUI.

Auto-refresh and compact view mode

All monitoring information should be updated periodically in the GUI. A “compact view” mode shall be available in order to present a simplified summary view of the detector partitioning status. This mode is to hide information about DAQ partitions, to which no part of the detector is assigned.

Trigger Monitoring

L1A rate and dead time information (TCS counters) must be monitored as well as the raw trigger bit rates (FDL “Final Decision Logic” counters). Information about down-scaling (“prescaling”) and masking of FDL triggers needs to be displayed, too.

Standalone control and monitoring of GT's state machine

The Global Trigger's logical state machine has several transition steps: initialization, partition dependent configuration and start/pause/resume/stop control for each partition. The state machine must be controllable from the GUI. This includes applying Trigger Supervisor Configuration and Trigger Supervisor Partition Configuration keys from the database. On the other hand, if the GT is controlled by the Central Cell, its state transitions must be traceable in the display.

Database interaction

A database population and modification panel should allow interacting with the Global Trigger configuration database, in particular, to browse the available configurations and to create new keyed setups in the database.

Expert-level hardware monitoring and control

An expert control panels should allow for a detailed overview of the hardware status. Another GUI is needed to allow expert control to set up individual hardware functions of the GT.

4.2.2 Non-functional Requirements

The GTCP should allow a comfortable setup and monitoring of the whole GT hardware. All the functionalities should be accessible in a minimum number of clicks (e.g. less than 3). Graphical status displays and partition assignment should be color-encoded and text-based. The Control Panel must be usable in an intuitive way. Features should be grouped according to functionality but also in view of typical usage. The GUI should be fast and compatible with standard browsers (Mozilla Firefox, Microsoft Internet Explorer, and Chrome). The event response time should be on the order of or less than a second. Loading time can be larger but must not exceed 10 seconds.

4.3 Architecture and Implementation

The general approach in the software is to build the GTCP on top of existing GT TS commands and monitoring infrastructure (i.e., like in the class structure diagram of Figure 2). These resources will be accessed using the TS framework and displayed using the AjaXell library.

4.3.1 Current Implementation Status

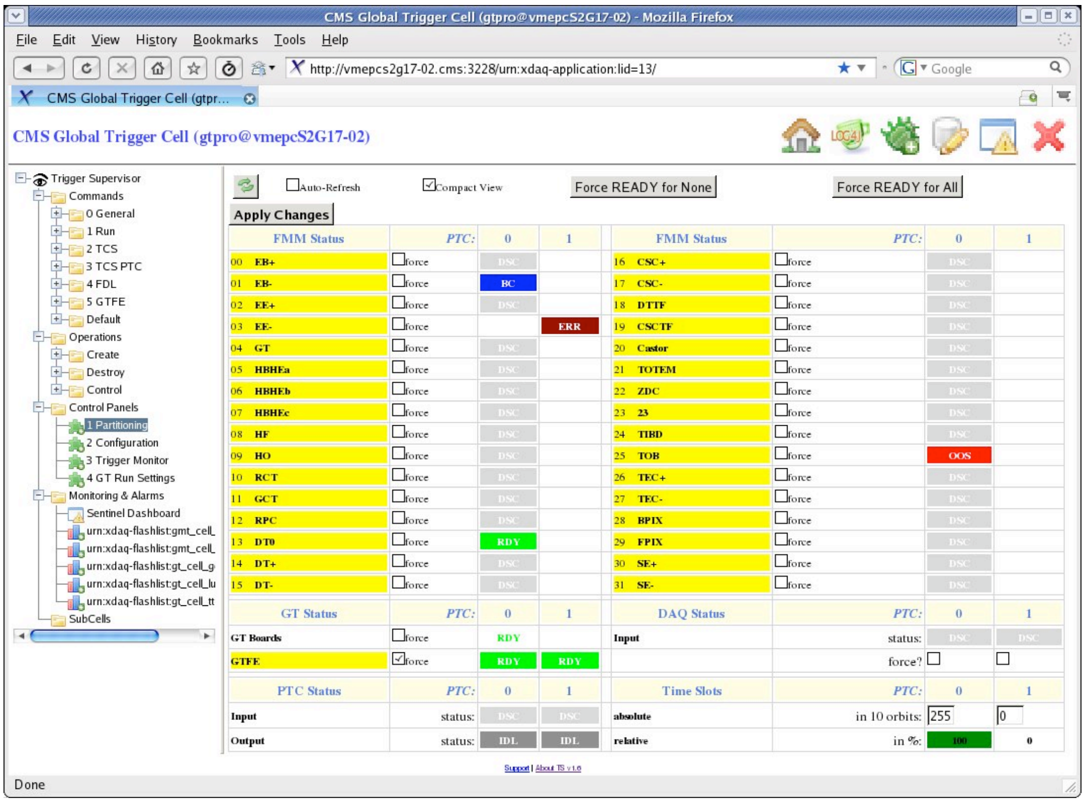
The Global Trigger Control Panel has entered the second implementation phase (for the first phase see [11]). Currently the implementation of the control panel includes, among others, the following features:

- Monitoring and control of status information
- Display and modification of the partitioning setup
- Trigger monitoring
- FDL trigger mask and prescaling control
- GUI access to FSM-steering commands

4.3.2 Implementation Example: The “Partitioning” GUI

A screenshot of the “Partitioning” GUI of the Global Trigger Control Panel is shown in Figure 8. It visualizes (in “Compact View” mode) an example TCS partitioning setup with two of the eight DAQ partitions in use:

- The real status of all TTC partitions is shown (none are forced “Ready”).
- Most detector parts are assigned to DAQ partition 0, only EE- belongs to partition 1. Among the appearing detector status codes are “DSC” (Disconnected), “BC” (Bad Code), “ERR” (Error), “OOS” (Out-Of-Sync), and “RDY” (Ready).
- None of the partitions are Running (output is “Idle”). Only partition 0 has a non-zero time slot assigned (255 times 10 orbits).


CMS Global Trigger Cell (gtpro@vmepcS2G17-02) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://vmepcs2g17-02.cms:3228/urn:xdaq-application:lid=13/
Google
CMS Global Trigger Cell (gtpr...
Trigger Supervisor
Commands
0 General
1 Run
2 TCS
3 TCS PTC
4 FDL
5 GTFE
Default
Operations
Create
Destroy
Control
Control Panels
1 Partitioning
2 Configuration
3 Trigger Monitor
4 GT Run Settings
Monitoring & Alarms
Sentinel Dashboard
urn:xdaq-flashlist:gmt_cell_
urn:xdaq-flashlist:gmt_cell_
urn:xdaq-flashlist:gt_cell_g
urn:xdaq-flashlist:gt_cell_lu
urn:xdaq-flashlist:gt_cell_tt
SubCells
Auto-Refresh
Compact View
Force READY for None
Force READY for All
Apply Changes
Support | About TS v1.6
Done

CMS Global Trigger Cell (gtpro@vmepcS2G17-02)

FMM Status	PTC:	0	1	FMM Status	PTC:	0	1
00 EB+	☐force	DSC		16 CSC+	☐force	DSC	
01 EB-	☐force	BC		17 CSC-	☐force	DSC	
02 EE+	☐force	DSC		18 DTTF	☐force	DSC	
03 EE-	☐force		ERR	19 CSCTF	☐force	DSC	
04 GT	☐force	DSC		20 Castor	☐force	DSC	
05 HBHEa	☐force	DSC		21 TOTEM	☐force	DSC	
06 HBHEb	☐force	DSC		22 ZDC	☐force	DSC	
07 HBHEc	☐force	DSC		23 23	☐force	DSC	
08 HF	☐force	DSC		24 TIBD	☐force	DSC	
09 HO	☐force	DSC		25 TOB	☐force	OOS	
10 RCT	☐force	DSC		26 TEC+	☐force	DSC	
11 GCT	☐force	DSC		27 TEC-	☐force	DSC	
12 RPC	☐force	DSC		28 BPIX	☐force	DSC	
13 DT0	☐force	RDY		29 FPIX	☐force	DSC	
14 DT+	☐force	DSC		30 SE+	☐force	DSC	
15 DT-	☐force	DSC		31 SE-	☐force	DSC	
GT Status	PTC:	0	1	**DAQ Status**	PTC:	0	1
GT Boards	☐force	RDY		Input	status:	DSC	DSC
GTFE	☑force	RDY	RDY		force?	☐	☐
PTC Status	PTC:	0	1	**Time Slots**	PTC:	0	1
Input	status:	DSC	DSC	absolute	in 10 orbits:	255	0
Output	status:	IDL	IDL	relative	in %:	100	0

Figure 8: Screenshot of the “Partitioning” Control Panel in the Global Trigger Cell (February 2009).

4.4 Performance

Tests have been carried out in order to measure the performance of the GTCP concerning the *total response time* of typical operations (time between clicking a button and the final response of the GUI when the corresponding task is finished). The same environment was used as specified for the tests in section 3.3.

The measurements refer to the Global Trigger online software as of February 19th, 2009. Table 1 summarizes the results.

GUI operation	*total response time (*ms*)*	*data transfer (kB)*
Click to display the "Parititioning" panel	~ 280 ms	~ 20
Click to apply changes in the "Partitioning" panel	~ 230 ms	~ 18
Click refresh button in the "Partitioning/Status" panel	~ 230 ms	~ 18
Click to display "Configuration" panel	~ 110 ms	~ 13
Click to display "Configuration" sub-panel "Random Trigger Setup"	~ 30 ms	~ 4
Click to display "Configuration" sub-panel "Algorithm Trigger Setup"	~ 550 ms	~ 53
Click to display "Configuration" sub-panel "Partition Mask Setup"	~ 120 ms	~ 8

Table 1: Performance measurement results: total response time for typical GUI operations

All response times are below one second, fulfilling the requirements stated earlier. The ratio of response time to data transferred is roughly 10 ms/kB, which is compatible with the measurements presented in Figure 6.

5 Conclusions

A software infrastructure has been provided for the Level-1 Trigger of the CMS experiment. This infrastructure allows the creation of GUI plug-ins for the TS control nodes. The development of these control panels allows the subsystems to migrate their standalone tools to a L1 Trigger homogeneous software system. If this is accomplished the learning curve of operators and hardware and software experts is reduced. Apart from the reduced learning time, the solution also reduces the cost and improves the quality of the software configuration management tasks for the L1 Trigger.

The implemented solution allows a compact, modular and layered implementation of control panels. The solution also assures an adequate responsiveness to the user and an efficient use of the network bandwidth. The usefulness has been demonstrated in the GT Control Panel case.

Currently, several L1 Trigger subsystems and CMS detectors have developed their custom control panels: Regional Calorimeter Trigger (RCT), Global Muon Trigger (GMT), Drift Tube Track Finder (DTTF), Hadronic Calorimeter (HCAL). There are also plans to start new control panels for the migration of the GT Trigger Menu GUI and the central control panel of the L1.

References

[1] The CMS Collaboration, S Chatrchyan et al, "The CMS experiment at the CERN LHC", 2008 JINST 3 S08004.
[2] The CMS Collaboration, The Trigger and Data Acquisition Project, Vol. I: The Level-1 Trigger, CERN LHCC 2000-038 (2000).
[3] The CMS Collaboration, The Trigger and Data Acquisition Project, Vol. II: Data Acquisition and High-Level Trigger, CERN LHCC 2002-26 (2002.
[4] I. Magrans, J. Varela, C.-E. Wulz, "Concept of the CMS Trigger Supervisor", IEEE Trans. Nucl. Sci. 53 (2006) 474-483.
[5] J. Gutleber, L. Orsini et al., "Hyperdaq, Where Data Adquisition Meets the Web", in Proc. of the 10th International Conference on Accelerator and Large Experimental Physics Control Systems, Geneva, Switzerland, 10-14 Oct. 2005.
[6] I. Magrans de Abril, M. Magrans de Abril, "Enhancing the User Interface of the CMS Level-1 Trigger Online Software with Ajax", 15th Real Time Conference, Fermi National Accelerator Laboratory in Batavia, IL, USA, May 2007.
[7] Miller, R. B., "Response Time in Man-computer Conversational Transactions", Proc. AFIPS Fall Joint Computer Conference Vol. 33, 267-277, 1968.
[8] A. Bouch, A. Kunchinsky and N. Bhatti, "Quality is in the eye of the beholder: meeting users' requirements for Internet quality of service" Proceedings of the SIGCHI conference on Human factors in computing systems, The Hague, The Netherlands, 2000.

[9] M. Jeitler et al, “The Level-1 Global Trigger for the CMS Experiment at LHC”, J. Instrum. 2 P01006, 2007.
[10] P. Glaser, “System Integration of the Global Trigger for the CMS Experiment at CERN”, Technical University of Vienna, March 2007.
[11] A.-J. Winkler, “Suitability Study of the CMS Trigger Supervisor Control Panel Infrastructure: The Global Trigger Case”, Vienna University of Technology, February 2008.