Available on the CMS information server **CMS NOTE 2011/007**

The Compact Muon Solenoid Experiment

CMS Note

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

2011/10/28

The Architecture of the CMS Level-1 Trigger Control and Monitoring System

Marc Magrans de Abril[1], Carlos Ghabrous Larrea[1], Josef Hammer[2], Christian Hartl[2], and Zhen Xie[3]

[1] University of Wisconsin, Department of Physics, US
[2] CERN, Physics Department, Switzerland
[3] Princeton University, Department of Physics, US

Abstract

The architecture of the Level-1 Trigger Control and Monitoring system for the CMS experiment is presented. This system has been installed and commissioned on the trigger online computers and is currently used for data taking at the LHC. It has been designed to handle the trigger configuration and monitoring during data taking as well as all communications with the main run control of CMS. Furthermore its design has foreseen the provision of the software infrastructure for detailed testing of the trigger system during beam down time. This is a medium-size distributed system that runs over 40 PCs and 200 processes that control about 4000 electronic boards. The architecture of this system is described using the industry-standard Universal Modeling Language (UML). This way the relationships between the different subcomponents of the system become clear and all software upgrades and modifications are simplified. The described architecture has allowed for frequent upgrades that were necessary during the commissioning phase of CMS when the trigger system evolved constantly. As a secondary objective, the paper provides a UML usage example and tries to encourage the standardization of the software documentation of large projects across the LHC and High Energy Physics community.

1 Introduction

The Compact Muon Solenoid (CMS) experiment is one of the two general-purpose particle physics detectors built for the Large Hadron Collider (LHC) at CERN, the European Organization for Nuclear Research [1]. Since billions of collisions occur each second and only a small fraction of these can be stored, events have to be selected online according to their properties. The trigger system is meant to reduce the data rate of 40 TB/s down to 100 MB/s, where the output limit is set to make best use of the available offline storage resources. Finally, the data is distributed and stored in the CMS offline computing system which is based on the LHC Computing Grid (see Figure 1) [2, 3]. The trigger system has been implemented in two consecutive stages (Levels). The Level-1 Trigger (L1) consists of custom-developed and largely programmable electronics, and the High Level Trigger (HLT) is a large computer farm [4, 5].

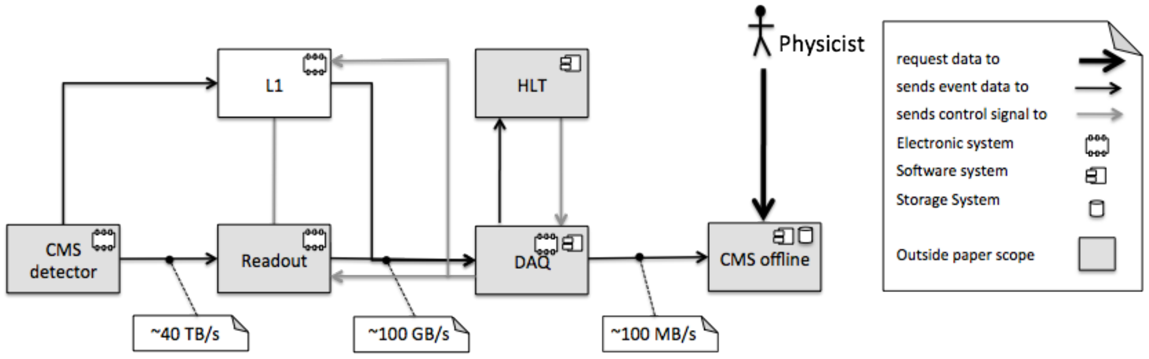

Figure 1: Diagram of the CMS event data flow. The boxes represent electronic, computing, or storage subsystems. The arrows always follow the causality direction. That is, the tail points to the subsystem starting the action while the head points to the subsystem reacting to it.

The objective of the present paper is two-fold. The primary objetive is to discuss and present the L1 Trigger Control and Monitoring System (L1CMS) architecture and its key aspects. The L1CMS is a medium-scale (i.e. 40 computers, 200 processes, 800 thousand lines of source code) distributed software system meant to configure, monitor and test the L1, and to provide appropriate human and machine interfaces for experts, shifters, and the Run Control and Monitoring System (RCMS) of the CMS experiment [6, 7]. The system has already been installed and commissioned, and it is being used for data taking. The description of its architecture has an interest per se as the description of a successful system, and as documentation for existing developers, operators, and managers to cope with the future maintenance and improvements.

The secondary objective is to demonstrate that a system architecture can be described without the help of metaphorical diagrams or pictures. This is, the authors advocate for the use of standardised diagram notations to describe software systems [8]. This paper will use a subset of UML (Universal Modelling Language) to describe the run-time structure and behavior and of the L1CMS [9, 10]. Although the authors want a notice that several standard notations will improve existing practices, they consider that amongst the different notations UML has the best trade-off between number of users, learning curve, and degree of formalisation. The compile-time structure of the system wont be described because this is already achieved by the automatic documentation tools (i.e. Doxygen, javadoc, etc.), and because it is on the run-time behavior where there are the main comprehension difficulties.

The paper is organized as follows. Section 2 describes the L1 system, its interfaces, and useful metrics about its size and complexity. Section 3 describes the L1CMS interfaces and its relation with its hardware counterpart. Section 4 focuses in the package structure (i.e. compile-time view). Section 5 to 7 present the architecture of the configuration, monitoring, and testing services. Section 8 presents the deployment process of the different software packages. Finally, Section 9 summarizes the work.

2 The Level-1 Trigger: Interfaces and Subsystems

The purpose of this section is to give an overview of the L1 hardware. This is going to be achieved by describing the relation of the L1 with the rest of the CMS experiment, and its decomposition in different subsystems.

The L1 is a custom hardware system meant to select those events with interesting physics signatures. This system is the first of a two phase selection process that reduces the data rate of the whole experiment to a manageable 100 MB/s.

Figure 2 shows the context diagram for the L1. That is, the highest-level view of the hardware system. This diagram represents the possible interactions of the L1 with its environment. The actors with which the L1 interacts and the causal origin of the interaction is also shown (e.g. the L1CMS always polls the L1, so it is never interrupt-driven by the L1).

The Figure shows five actors interacting with the L1: CMS detector (i.e. sends event-data), Readout (i.e. receives the accept signal and the clock, and sends its buffer status), LHC (i.e. sends the clock), Data Acquisition System or DAQ (i.e. receives the intermediate state of the L1 as part of the event data), and the L1CMS (i.e. the means to configure, monitor and test the L1). Table 1 in Appendix A shows the description and type for each interface.

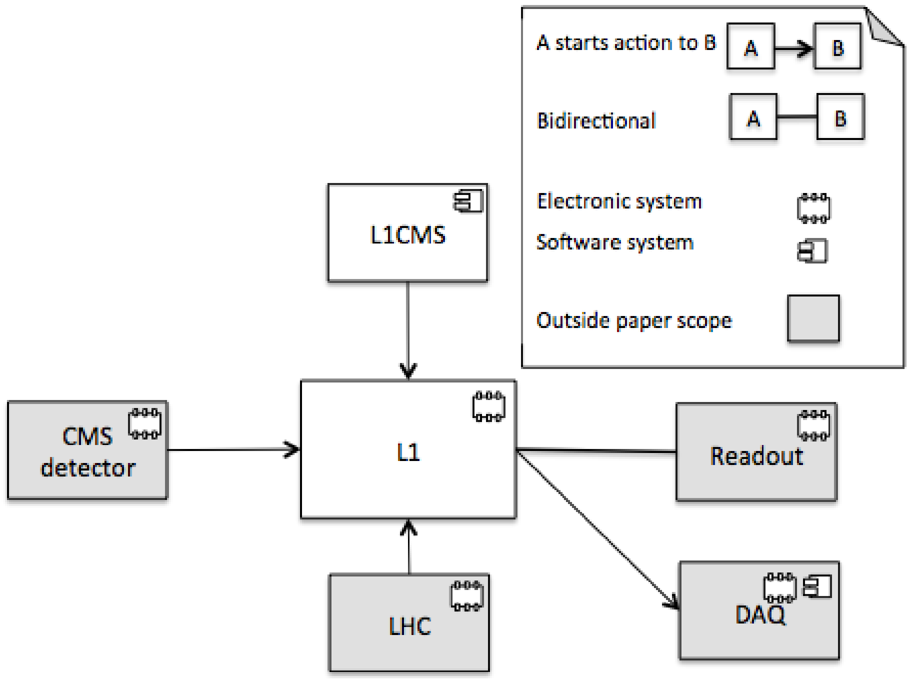

Figure 2: Context diagram of the L1. The diagram shows the L1 external interfaces. The arrows follow the causality direction. A line without arrowhead means that both subsystems can start the action (e.g. the L1 send the accept signal to the Readout and the Readout sends to the L1 its buffer status).

On the other hand, Figure 3 shows the component diagram of the L1. The external interfaces are shown, however the arrows now point to the internal L1 subsystems. The Figure also shows for completeness the internal flow of the event-data that is step-wise transformed into trigger objects, and processed hierarchically. Therefore, the L1 can be decomposed into 13 subsystems of different sizes and complexity (see metrics in Table 3 Appendix C). The inputs of the L1 are the event data from the CMS detector, the Trigger Throttling System (TTS) signal from the Readout, and the LHC clock. The outputs are the L1A signal and the intermediate trigger objects of the L1 computation.

The distribution of the clock and Trigger Timing and Control (TTC) signals has been removed to avoid cluttering the diagram. The Figure neither shows the PCI-VME interface from each L1 subsystem to the L1CMS for the same reasons. Finally, is should be noticed that although the Electromagnetic and Hadronic Calorimeter Trigger Primitive Generators (ECAL TPG and HCAL TPG), the Cathode Strip Chamber Local Trigger (CSC LT), and the Trigger Throttling System (TTS) are part of the L1, its control and monitoring software is out of this paper scope (i.e. in grey). The reason of this asymmetry is that the software controlling their electronics has been developed outside the L1 team, and it has been integrated as part of the ECAL, HCAL, CSC, and DAQ systems, respectively.

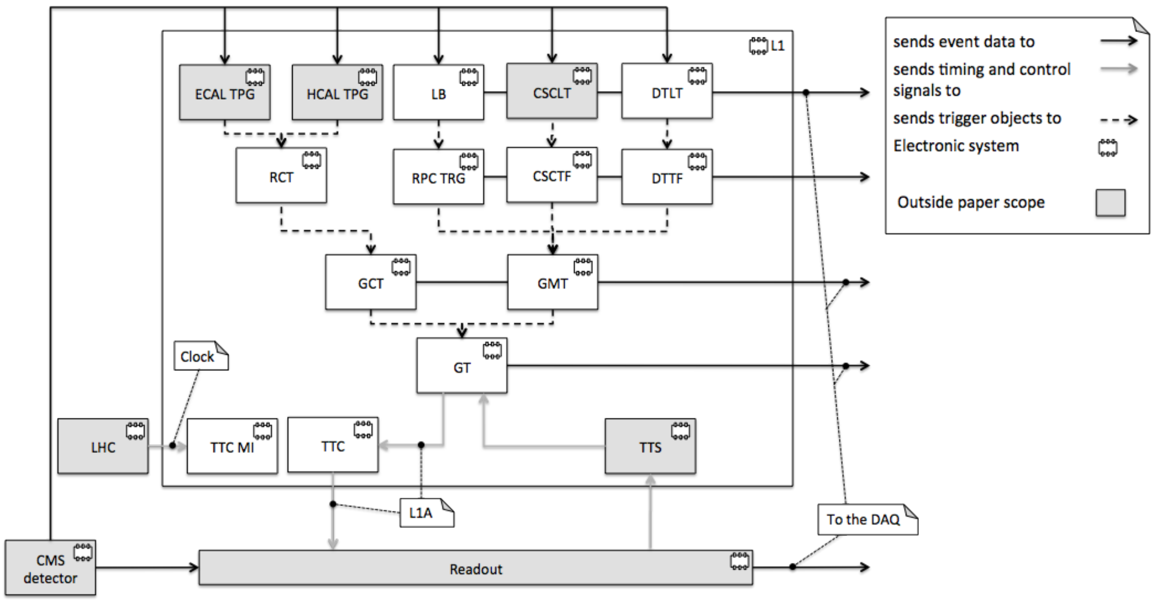

Figure 3: Component diagram of the L1 system. The diagram shows the L1 decomposition in 13 different subsystems and the interfaces involved in the event selection. The distribution of timing and control signals has been obviated to reduce the cluttering. The arrows direction follows causality relation.

3 The Level-1 Trigger Control and Monitoring System: Interfaces and Subsystems

This section is meant to provide the highest-level view of the L1CMS. That is, its external interactions with subsystems and people.

Figure 4 shows the L1CMS context diagram. The diagram shows the different actors interacting with the L1CMS and the causal origin of the interaction. The head of the arrow points to the subsystem providing the interface, while the tail touches the subsystem initially requiring it. If there is no arrow, a bi-directional interface is implied and both ends can start the interaction.

Table 2 in Appendix A further explains the interface characteristics and its relation with the services offered byt the L1CMS (i.e. configuration, monitoring, and testing).

The following **key aspects** should be noted from the context diagram and the interfaces Table:

- *Oracle Database (DB) is the configuration and monitoring DB backend*. The CERN Information Technologies department provides access to and management of an Oracle DB. Therefore, using Oracle is the most cost-effective way to access to persistency services.
- *Web services based architecture*. All the interfaces are web service based except those for Online-to-Offline (O2O), L1 Emulators, Web Based Monitoring (WBM), and of course, the L1 electronics (non-interrupt driven commands through PCI-VME bridge).
- *Query WBM DB to retrieve Luminosity data*. WBM provides a variety of CMS configuration and monitoring data through an HTTP Graphical User Interface (GUI), and also through public Oracle tables and views [11].
- *UNIX pipes over Secure Shell (SSH) to interface O2O and L1 emulator*s. This was an ad hoc solution to the problem of integrating a CMSSW application in the L1CMS [12].

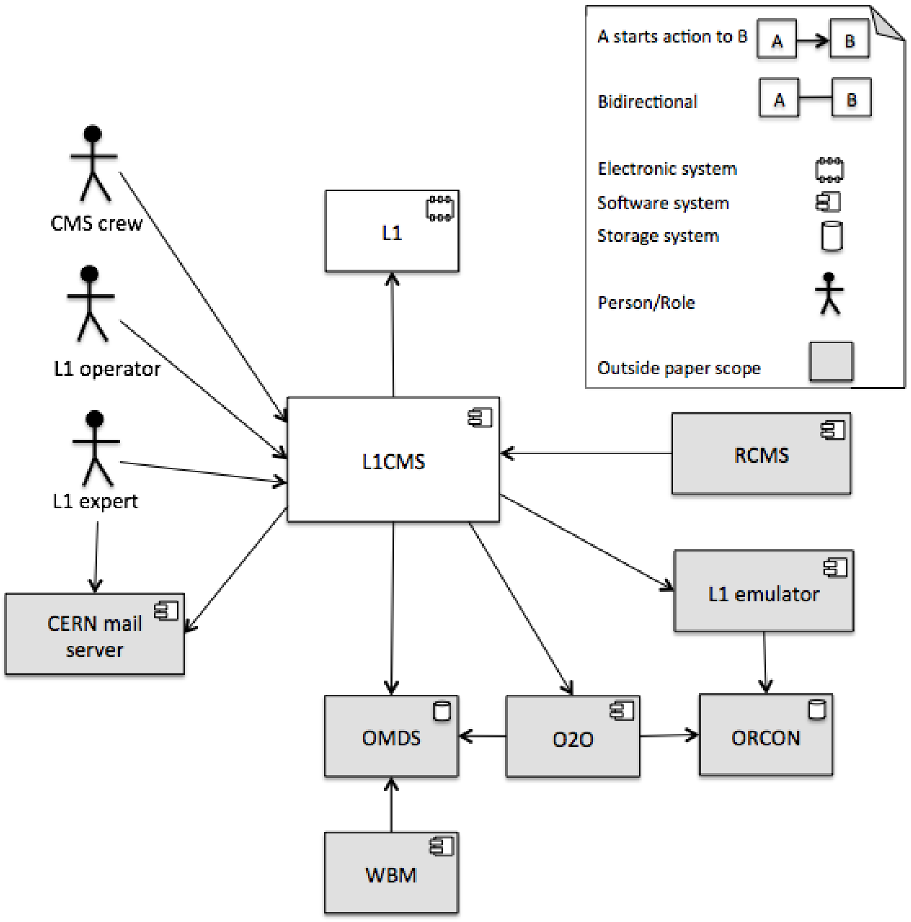

Figure 4: Context diagram of the L1CMS. The diagram shows the L1 external interfaces. The arrows follow the causality direction. A line without arrowhead means that both subsystems can start the action.

4 Package Structure

Figure 5 shows the compile-time dependencies between components of the L1CMS. In the upper part of the Figure the different L1CMS subsystem packages are shown. These packages correspond to the L1CMS subsystems software top-layer. The diagram also shows three different product lines. First, this figure shows the core control and monitoring packages that are built on top of C++, Cross-Platform Data Acquisition System (XDAQ), and Trigger Supervisor (TS) [13][6]. On the second hand, there is the Level-1 Function Manager (TRG FM), the top node of the L1CMS. That is a J2EE servlet based on the RCMS framework that runs on Apache Tomcat. The basic function of the TRG FM is to match interfaces between the RCMS Level-0 Function Manager (L0 FM aka the top node of the experiment control system) and the rest of the L1CMS. Finally, the L1 Trigger web Page (L1Page) is the third product line running also as an Apache Tomcat servlet. This one does not depend on RCMS and it is meant to be a browsable entry point for operator and experts.

Figure 5 also shows two types of run-time dependencies (i.e. `uses` relation). The first one is the Simple Object Access Protocol (SOAP) control chain from the L0 FM to the leafs of the L1CMS. The second run-time dependency is derived from the contract between the Tomcat J2EE container, and the L1 Page and the TRG FM servlets.

The following **key aspects** should be noted from the package structure diagram:

- *Scientific Linux version 4 (SLC4) as Operating System (OS).* Apart from the historical preference for open source solutions, CERN provides in-house support to SLC4 (and its future variants).
- *XDAQ and C++ on the leaf nodes of the distributed system.* Using C/C++ simplifies the software development of the hardware access layer. The driver and the first software layer are usually developed in C or C++ for performance and historical reasons. Therefore, using the XDAQ middleware, which is based on C++, simplified the integration with the hardware access layer. An additional advantage of using XDAQ (with respect to other C++ server containers) is that it also have in-house support.
- *XDAQ and C++ on the intermediate nodes of the distributed system.* Each of the L1CMS subsystems is a distributed system of his own that should also work standalone in the electronics laboratory or the developer testbed. Using a single technology (i.e. TS over XDAQ and C++) allowed a simplification of the software developer cycle. This is of much help in an environment with reduced manpower dedicated to control and monitoring software, with a high turn-over (30 to 50% in the last years), and with a high proportion of novices versus experts developers (30 to 100%). In fact, it is not unusual situations where the hardware experts had to work on software, or where an undergraduate is fully responsible of a subsystem software. This is, the decision of using XDAQ and C++ in the bottom nodes combined with the overall maintenance cost pushed the decision trade-off to the XDAQ/C++ side.
- *RCMS, Tomcat, and Java in the top nodes of the distributed system.* Usually, the top nodes of the a control system do not access to the hardware directly. In addition, for historical reasons RCMS/Tomcat/Java were going to be used as top nodes of the CMS control system. Therefore, the L1CMS was forced to comply with the RCMS SOAP interfaces which is enormously simplified (given that the SOAP interface is not stable) if we use the RCMS framework. In addition, the use of Java (against C/C++) and the additional heterogeneity is compensated by the excellent documentation of

the Java software stack, the large number of free libraries and tools, the automatic garbage collector, and the fast enough performance. In fact this is the reason why the Level-1 Page (L1Page, the entry point for the L1 information) was based on Tomcat and Java (but not in RCMS).

- *TS layer on top of XDAQ*. The TS Framework was developed to simplify the integration of the different nodes of the L1CMS. The TS constraints the L1CMS subsystem nodes to an homogenous SOAP interface, Finite State Machine (FSM) model, and GUI library. These features simplify the integration of the different subsystems (each one developed by a different institution) and reduces the maintenance cost.

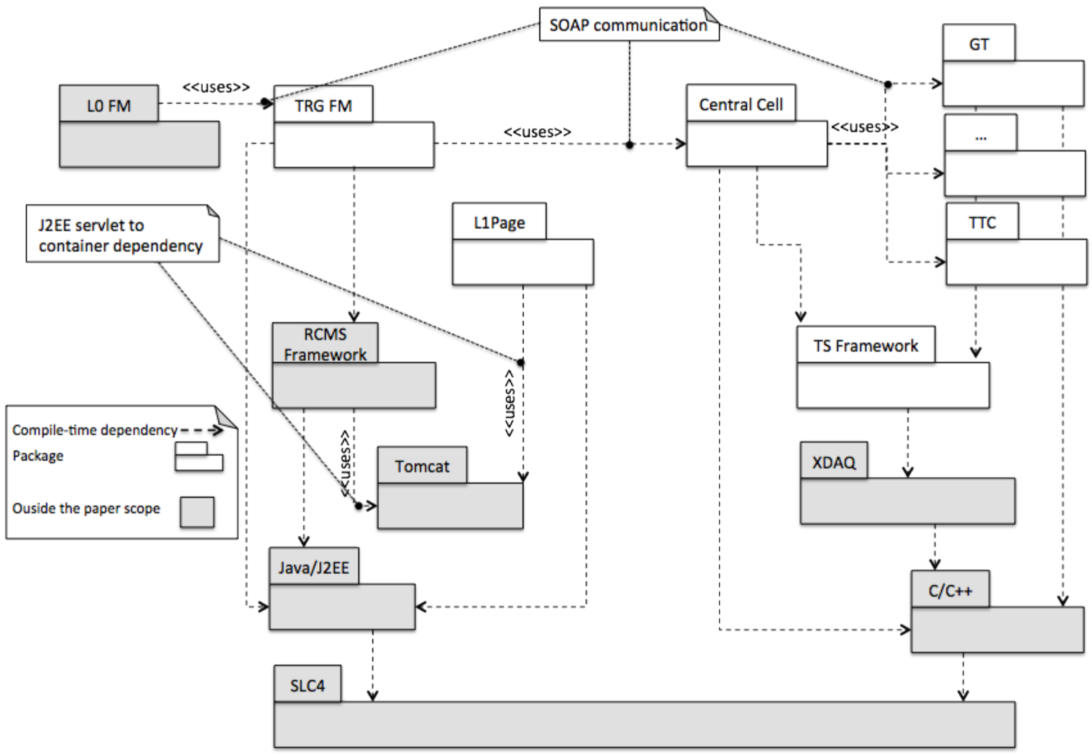

Figure 5: Package diagram of the L1CMS. All the dependencies shown are at compile-time except the ones related to Java servlet-to-container contract and to the SOAP communication.

5 Configuration Service

The L1CMS configuration service is meant to set the internal state of the L1 according to the experimental goals of the CMS experiment. This service establishes the configuration of the L1 from a unique key and assures that the process is reproducible and traceable. That is, once the configuration data is stored and linked to the configuration key, the operator is able to reproduce the same L1 state as many times as needed. If the process fails, then the failed attempt could be audited post-mortem not just analyzing the usual logs, but also taking into account the expected hardware configuration data.

This section is meant to describe the configuration service using two different views. The first view is a component diagram with the runtime dependencies and associated interfaces. The second view is the hierarchical activity diagram that models the different steps to configure and start the L1.

Figure 6 is a component diagram of the subsystems involved in the configuration process. The arrows represent the required (beginning) and provided (ending) remote interfaces and their types. The diagram refines the information described already in figure 3. The configuration follows a hierarchical topology from the RCMS to the subsystems. The diagram can be used also to backtrack the experiment faults to the possible source of the error, or to foresee the impacts of a change while following the causal path of the configuration process.

There are five components worth mentioning:

- *TRG FM (Level-1 Trigger Function Manager).* The TRG FM is meant to match the remote interfaces between RCMS and Trigger Supervisor (TS) frameworks.
- *Central Cell.* The Central Cell coordinates the interplay between L1 subsystems.
- *O2O.* The O2O process is meant to transfer the configuration data between the online and offline DBs.
- *WBM.* The Global Trigger (GT) retrieves the luminosity data from WBM DB in order to select the correct prescale set.
- *L1CE (Level-1 Configuration Editor).* The editor is a web page that simplifies the creation and maintenance of configuration data.

On the other hand, Figure 7 shows a hierarchical activity diagram of the CMS configuration process. The diagram represents the three steps needed to start the CMS data taking (i.e. pre-configure, configure, and start), and the one to stop it. As the diagram shows, the L1 is configured in two steps instead of just one. This peculiarity allows the parallel configuration of CMS once the clock distribution is set up in the Trigger Timing and Control Machine Inteface (TTC MI) and the Global Trigger (GT) (i.e. preconfigure activity). The diagram further refines the causal and time dependencies between subsystems during configuration. Therefore, it further simplifies the location of errors and the consequences of high-level changes.

The following configuration sequence can be derived from the lecture of both Figures:

1. The L1 operator or expert creates the configuration key using the L1 Configuration Editor (L1CE) from a web browser.

2. Also from a web browser, the CMS operator selects the configuration key and executes the `pre-configure` command from the L0 FM GUI.

3. RCMS propagates the `pre-configure` request to the TRG FM through SOAP.

4. The TRG FM executes an O2O command through SSH (Secure Shell) in order to set the interval of validity (IOV) for the configuration data, and if necessary, O2O also copies the data to the ORCON DB. In parallel, the TRG FM propagates the `pre-configure` command to the Central subsystem.

5. The Central subsystem propagates the `pre-configure` request as a `configure` request. First to the TTC Machine Interface (MI) and then to the Global Trigger (GT). After this action, the clock and synchronization signals are set up for the whole experiment.

6. As the clock and synchronisation signals are ready, the CMS operator can configure the rest of the experiment. Therefore, when the operator executes `configure`, the RCMS sends in parallel the `configure` request to the TRG FM and the rest of the experiment.

7. The TRG FM propagates the `configure` request in parallel to the rest of the L1CMS. The configure activity finishes once all the L1CMS subsystems are configured and the O2O

command (that started in `pre-configure`) is finished.

8. Once all the experiment subsystems are configured, the CMS operator executes the `start` command. This command first starts the rest of the experiment and finally the L1.

9. The `start` request is propagated to the rest of the L1CMS subsystems in parallel except for GT, which is the last to be started. Once this is done CMS starts the data-taking process.

10. In order to finish the data taking, the CMS operator will execute `stop`. This will propagate the `stop` request exactly in the reverse order that during `start`. First stopping GT, then the remaining L1 subsystems, and finally the rest of the experiment.

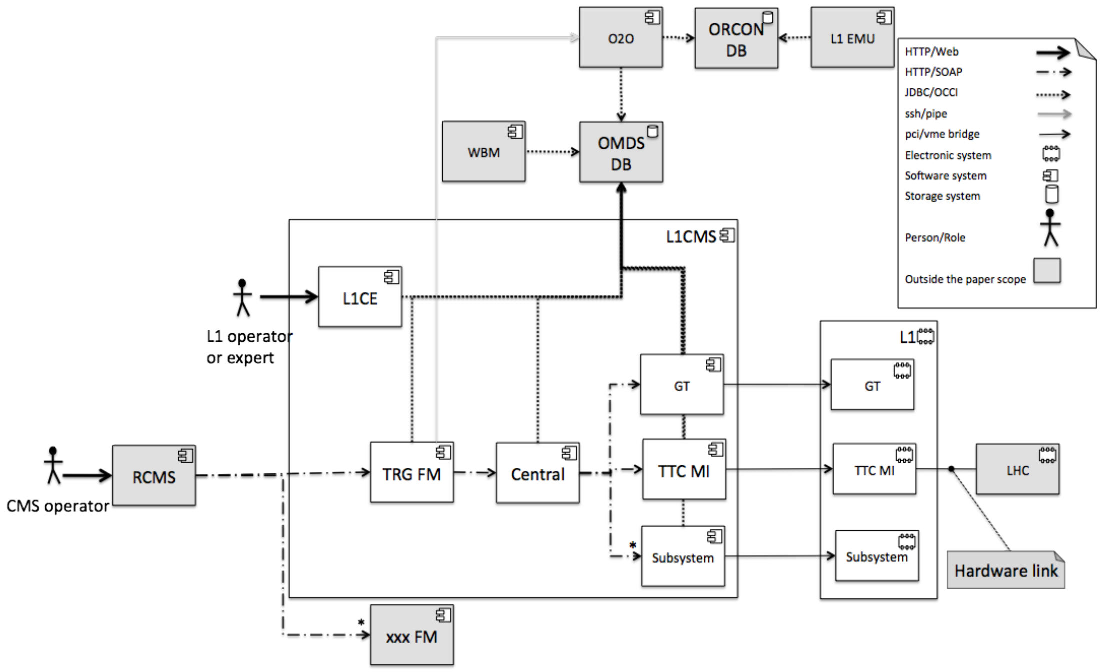

Figure 6: Component diagram of the L1CMS and its run-time dependencies during the configuration process. The arrows follow the causality direction (e.g. the CMS operator sends an HTTP message to the TRG FM).

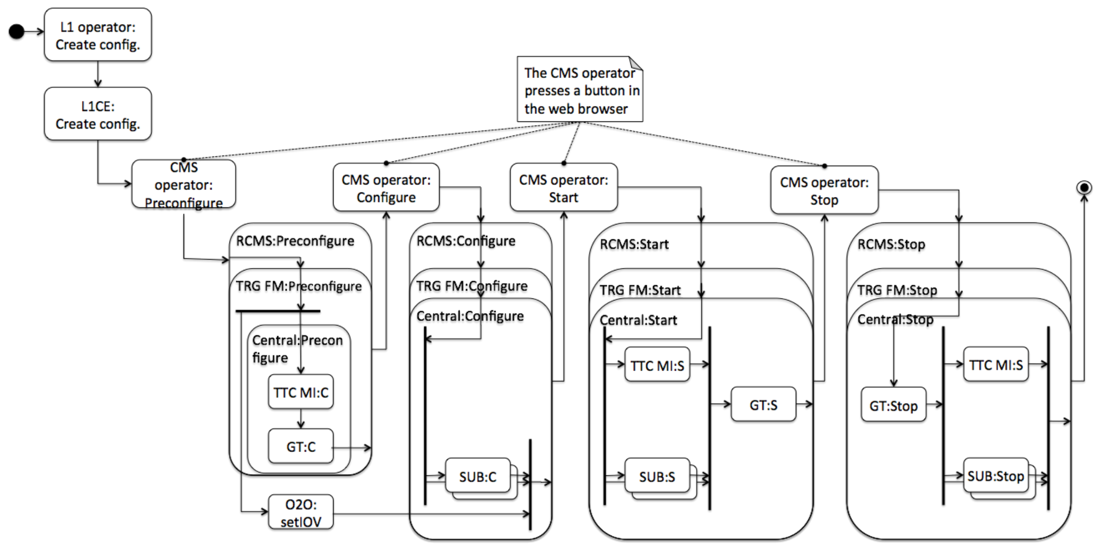

Figure 7: Activity diagram of the configuration process of the L1CMS and its subsystems. The arrows show the temporal sequence of the different configuration activities. The activity labels identify the subsystem and the activity name (i.e. `subsystem:activity`). The following abbreviations apply to the different activities: `P=Preconfigure`, `C=Configure`, `S=Stop` or `Start`.

6 Monitoring Services

The monitoring services are meant to report on the state of the L1 hardware and software. They are also meant to notify immediately to operators and experts about errors. The notifications are either shown as a message displayed in the L1 Page, as plot in WBM, or through a direct mail to L1 experts.

The architecture of the montoring system is described in two views using component diagrams: hardware monitoring and software monitoring.

Figure 8 shows the run-time dependencies associated to the monitoring of the L1 hardware. In case there is a hardware error, then a summary of the error is displayed in the L1 Page, notified by mail to experts, and if further details are needed, the operator or expert can check the subsystem specific web pages [14]. The information flows in the reverse order than in the configuration service. That is, the information is propagated from the L1 subsystems to the Central one through the XDAQ Monitoring and Alarming System (XMAS) [15]. Additionally, the hardware monitoring also comprises the monitoring of the GT rates, where the data is send by GT to OMDS, and then it is retrieved and presented by WBM.

As a special case of hardware monitoring, Figure 8 shows the dependency path meant to display the result of the configuration process in the L1 Page. That is, the TRG FM writes the configuration result that was forwarded to RCMS in a NFS file. Then, the file is opened and displayed by the L1 Page. This gives a seamless experience to the operator who does not need to have knowledge on the different tools to retrieve the different types of information.

On the other hand, Figure 9 shows the run-time dependencies associated to the monitoring of the L1CMS itself. The self-monitoring is achieved by periodic checks of the critical services availability through their HTTP API. If a service is unavailable, then an error appears in the L1 Page, and the experts are notified through mail. In that case, the operator is also instructed to restart the corresponding service from the L1 Page (i.e. clicking on a button that will restart several services in different machines through SSH).

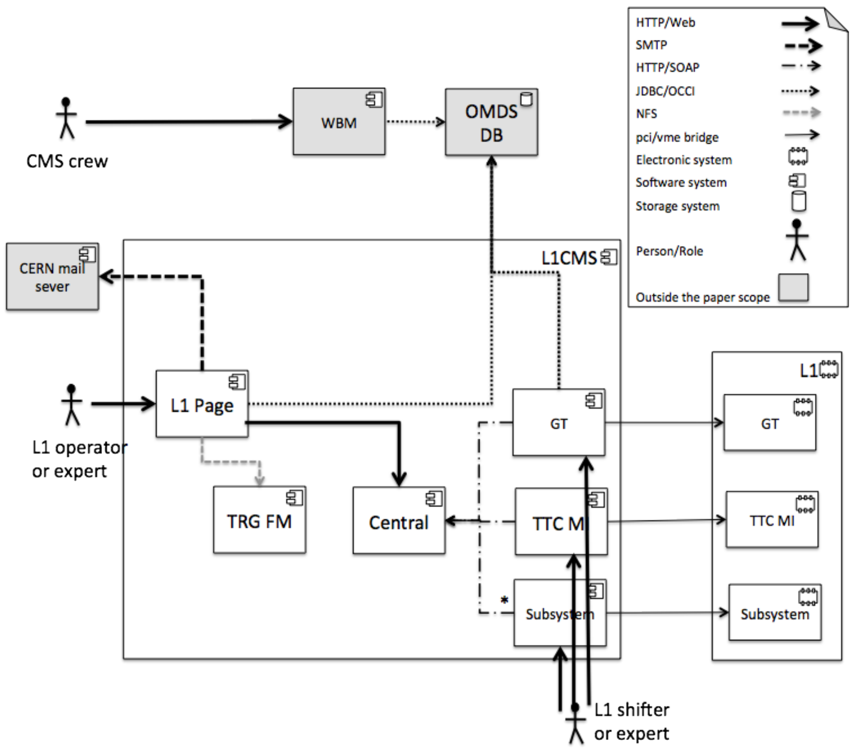

Figure 8: Component diagram of the hardware monitoring services and its run-time dependencies. The arrows follow the causality direction (e.g. the CMS operator sends an HTTP request to the L1 Page to get the state of the L1 subsystems).

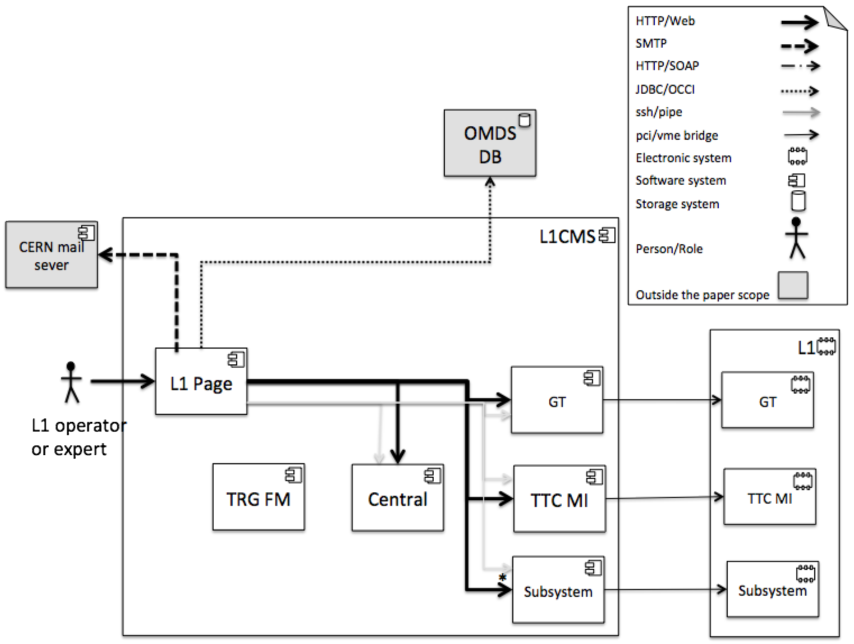

Figure 9: Component diagram of the process monitoring service and its run-time dependencies. That is, the monitoring of the L1CMS by the L1CMS itself. The arrows follow the causality direction (e.g. the L1 operator sends an HTTP request to the L1 Page to know if all the critical services are reachable, if they are not send another HTTP request to start the services via SSH).

7 Interconnection and Pattern Test Service

The Interconnection and Pattern Test (IPT) service is a stateful web service wit the objective to allow for the creation and execution of hardware tests between and within L1 subsystems. This service is meant to be used during the whole experiment lifecycle in order to assure the quality of fixes and upgrades.

The IPT architecture is similar to the one depicted in Figure 6 except for five aspects. First, the IPT is launched from the Central Cell, and it is not meant to be a CMS-wide service. Second, there is not just one IPT service, but many (one per inter- or intra-subsystem test). Third, the IPT activity diagram, and therefore its SOAP interface, is different from the one of the configuration service. As shown in Figure 10, there is a loop to exercise different hardware connection paths (potentially using different bit patterns each), sparing the need to set up the test again for each of the paths/patterns (i.e. repeat the `configure` activity, which takes O(100) seconds). Fourth, the orchestration of subsystem activities is defined as part of the IPT configuration data (i.e. it is not fixed like in the configuration service).This facilitates the customization of tests. Finally, the L1 emulators could be accessed as part of the `setup` transition in order to create the input and output patterns through a system call (i.e. calling a remote script via SSH).

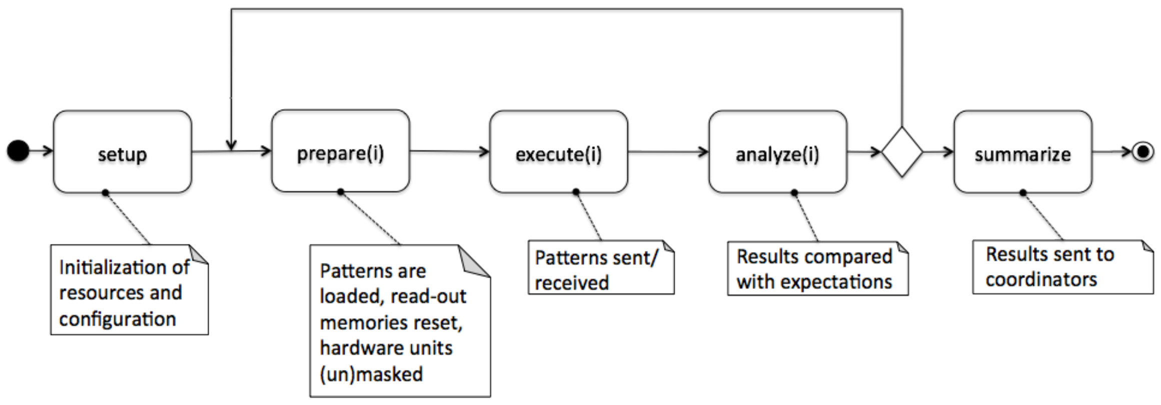
setup
prepare(i)
execute(i)
analyze(i)
summarize
Initialization of resources and configuration
Patterns are loaded, read-out memories reset, hardware units (un)masked
Patterns sent/ received
Results compared with expectations
Results sent to coordinators

Figure 10: Activity diagram of the IPT service running in each node.

8 Deployment Process

The deployment process is the set of activities needed to deploy the software artifacts to its final location (i.e. the package source code, the compiled binaries, or an RPM containing the binaries and headers), and therefore to make the software system available for use.

This becomes an important part of any system when the software has to be deployed in more than a couple of machines, or it has multiple processes. The L1CMS is comprised by 40 PCs and 200 processes connected to hardware via PCI to VME optical links (see Table 3 in Appendix C). In addition, the LqCMS subsystems should be installed in several hardware and software facilities around the world. Therefore, the manual deployment to all this machines is not an option.

Figure 11 shows the deployment diagram for the L1CMS in the experiment network. The diagram shows the movement of artifacts for each of the packages presented in Section 4. This is, the diagram shows how source files, scripts, and binary executable files are moved from the software repository to the production site.

The deployment in the testbeds around the world has been removed to simplify the diagram. Nevertheless, the deployment is done using the XDAQ and TS YUM repositories executing a single command in the shell.

Basically, the Figure shows two different deployment scenarios:

- *Quattor deployment using RPMs (XDAQ, TS, RCMS, TTC, and L1 Page packages)*. This method is the preferred one.There is an on-going effort to move the rest of the L1CMS to this deployment method. In that case, the RPM (RPM Package Manager formerly Redhat Package Manager) containing binaries and headers are created outside the experiment network and tested. Then, the RPMs are sent to the Quattor server, and the server installs them automatically to all the required hosts [16]. This procedure simplifies the maintenance of large PC farms. There is a small difference in the deployment process between XDAQ and TS packages in one side, and the RCMS in the other. The former RPMs are previously stored in a YUM repository and therefore the developers can test the RPM packaging before deployed in production. This is not the case for the RCMS framework.
- *Build and deployment using NFS (TRG FM and L1 Page)*. The source code is checked out from CVS and build inside the experiment network on a NFS (Network Files System) mounted PC. Then the binaries are accessible from any PC. This method is

simple and easy to understand for newcomers. That is, newcomer's workflow do not change with respect to the usual work experience with single machine compilation and deployment. The distributed file system allows the access to the same binaries everywhere in the experiment cluster although they have been compiled in a single place (as far as the same OS is installed in all the machines). Unfortunately, using this deployment method it is not possible to track the artifacts version, nor to rollback without expert intervention if something goes wrong.

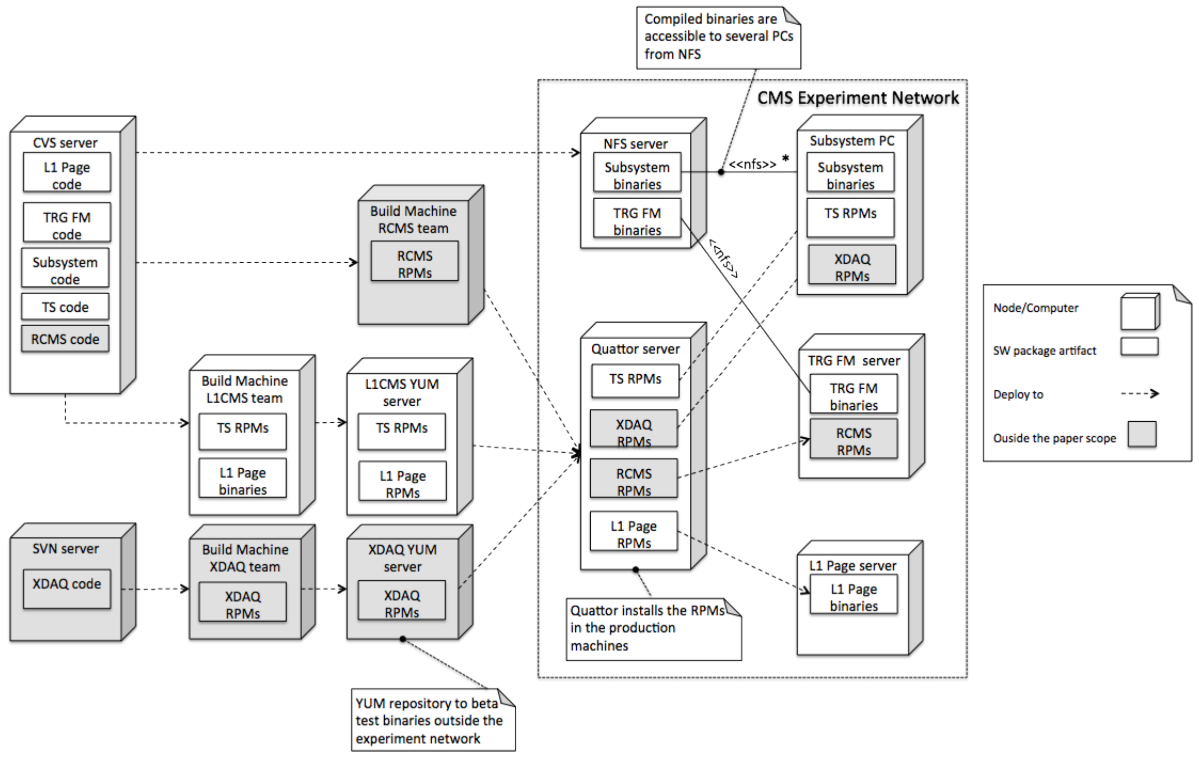

Figure 11: Deployment diagram of the L1CMS system. The servers and steps involved in the deployment process are shown. The dashed arrows follow the causality direction. Except for the software subsystem packages (TRG FM included) which are deployed in NFS, the software is deployed as RPMs using Quattor.

9 Summary

The present paper describes the architecture of the L1CMS and the rationale behind its key characteristics. The relation of the software with the hardware, the internal and external interfaces, and the decomposition in subsystems and services has been shown. The authors have also made a deliberate attempt to visualize the information using UML. Surprisingly enough, this effort has also helped the authors to reduce the size of the paper. A posteriori, this seems obvious because the syntactic and semantic content of the diagrams and notation used can be found in the corresponding standards and manuals. The authors expect that such an attempt will simplify the understanding of the L1CMS system for newcomers, and its evolution. Hopefully, the attempt will also encourage the LHC and High Energy Physics community to improve its software documentation by using standard notations for its software systems..

Acknowledgements

The authors would like to thank all the L1 subsystem experts for their feedback regarding the number of crates and boards in Appendix C. We thank Gian Piero di Giovanni for his review on the first version of the note. We thank Vasile Ghete for his detailed explanations of the function and structure of the L1 emulators. Finally, we thank Giuseppe Codispoti for his helpful insight on various components of the CMS software stack.

References

[1] CMS Collaboration, "The CMS experiment at the CERN LHC", *J. Instrum.* **3** (2008).

[2] The LCG TDR Editorial Board, "LHC Computing Grid", technical report, CERN, 2005.

[3] CMS Collaboration, "The CMS Computing Project", technical report, CERN, 2005.

[4] CMS Collaboration, "CMS TriDAS project: Technical Design Report; 1, the trigger systems", technical report, CERN, 2000.

[5] CMS Collaboration, "CMS trigger and data-acquisition project : Technical Design Report", technical report, CERN, 2002.

[6] I. Magrans et al., "Concept of the CMS Trigger Supervisor", *IEEE Trans. Nucl. Sci.* **53** (2006) 474–483.

[7] A. Petrucci et al., "The Run Control and Monitoring System of the CMS Experiment", in *J. Phys.: Conf. Ser.*, volume 118. 2008.

[8] R. Wieringa, "A survey of structured and object-oriented software specification methods and techniques", *ACM Comput. Surv.* **30** (December, 1998) 459–527. doi:http://doi.acm.org/10.1145/299917.299919.

[9] D. Pilone and N. Pitmanoks, "UML 2.0 in a Nutshell". O'Reilly Media, 2005.

[10] J. Garland and R. Anthony, "Large-Scale Software Architecture: A Practical Guide using UML". Wiley, 2002.

[11] W. Badgett et al., "CMS Web-Based Monitoring", in *IEEE/NPSS Real Time Conf.*, volume 17. 2010.

[12] V. Innocente, L. Silvestris, and D. Stickland, "CMS Software Architecture", *CMS Note* **2000/47** (2000).

[13] J. Gutleber et al., "Hyperdaq, Where Data Adquisition Meets the Web", in *Proc. Intl. Conf. Accel. and L. Exp. Phys. Control Sys.*, volume 10. 2005.

[14] M. Magrans de Abril et al., "Homogeneous User Interface Infrastructure for Expert Control of the Level-1 Trigger", *CMS Note* **2010/05** (2010).

[15] G. Bauer et al., "Monitoring the CMS Data Acquisition System", in *International Conference on Computing in High Energy and Nuclear Physics*, volume 17. 2009.

[16] M. Garca Leiva, R. et aland Barroso Lpez, G. Cancio Meli, B. Chardi Marco et al., "Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics", *Journal of Grid Computing* **2** (2004) 313–322.

A External interfaces of the Level-1 System

Table 1 describes the different external interfaces of the L1. The Table extends the information provided in Section 2.

Table 1: Summary of the L1 external interfaces

Subsystem	Description	Type
CMS detector	Event data is sent to the L1 calorimeter Trigger Primitive Generators (TPG), the muon local triggers, and the Resistive Plate Chambers (RPC) pattern comparator	Optical, subsystem specific
DAQ	Intermediate computation steps of the L1 decision are sent to the DAQ in order to compare them with the L1 emulators	Serial Link
Readout	Readout buffer status is sent through the Trigger Throttling System (sTTS) to the L1 to stop it sending L1A If the event is selected, then the L1 sends a Level-1 Accept (L1A) signal to the Readout electronics to deliver it to the DAQ. If the L1A does not arrive in less than 3.2 s the event is rejected Clock and control signals are sent from the L1 to the Readout electronics through the TTC	Trigger Throttle System (TTS) link Trigger Timing and Control (TTC) Link TTC link
L1CMS	Configuration, monitoring and testing commands are send to the L1	PCI-VME bridge, subsystem specific
LHC	Orbit and clock signals are received by the TTC Machine Interface (TTC MI) subsystem and distributed to all the experiment subsystems	Optical fiber

B External interfaces of the Level-1 Control and Monitoring System

Table 2 describes the different external interfaces of the L1CMS. The Table extends the information provided in Section 3.

Table 2: Summary of the L1CMS external interfaces and its relation with the different L1CMS services

Subsystem	Description	Service	Type
L1 Expert	Creates configuration data stored it in the configuration DB	Configuration	HTTP/Web Browser
	Performs tests through a GUI and check the results	Testing	HTTP/Web Browser
	Monitors the system while running or testing	Monitoring	HTTP/Web Browser
L1 operator	Assures the correct operation of the L1 thorough the use of the L1CMS monitoring services	Monitoring	HTTP/Web Browser
	Fine tunes the L1 configuration according to the experiment needs (e.g. prescales, random triggers, etc.)	Configuration	HTTP/Web Browser
	Restarts an L1CMS process if it dies or hangs	Monitoring	HTTP/Web Browser
CMS crew	Monitors the system while running	Monitoring	HTTP/Web Browser
CERN mail server	If the monitoring system detects an error, then a mail notification is send to a mailing list to be subscribed by L1 experts	Monitoring	SMTP
L1	Sends and receives data from the L1 electronics in order to configure, monitor, and test (pollint always, no interrupt-driven)	Configuration, Monitoring, Testing	PCI-VME bridge
OMDS DB (aka Configuration and Monitoring DB)	The OMDS DB is the configuration and monitoring DB for the CMS experiment. It is a set of Oracle schemas used to store configuration, monitoring, and testing data	Configuration, Monitoring, Testing	JDBC, OCCI
O2O (OMDS to ORCON)	The O2O process transfers configuration data from the online configuration DB (aka OMDS) to its offline counterpart (aka ORCON). The L1 emulators, the HLT, and the CMS analysis and reconstruction software use the configuration data in ORCON	Configuration	Unix pipes over SSH
RCMS	RCMS is the experiment control system. This system sends SOAP commands to the top node of the L1CMS to execute a given Finite State Machine transition. Informative and error messages are reported back	Configuration	SOAP over HTTP
L1 Emulator	The L1 emulators are used to assure the correct behavior of the L1 during data taking. This is, they compare the input and output objects of various L1 subsystems versus their expected value. They are also used to generate the input and output patterns to test the L1. Finally, the GT emulator is also used by the HLT to provide the list of L1 objects and algorithms that actually fired in each event	Testing	Unix pipes over SSH
WBM (Web Based Monitoring)	The web based monitoring service accesses the L1CMS data on OMDS, and creates a copy in their own DB schema	Monitoring	JDBC, OCCI
	Luminosity System data is also accessible from WBM schemas in OMDS. The Global Trigger uses this data in order to select the correct prescale set	Configuration	JDBC/OCCI

Continues on next page...

. . . Continued			
Subsystem	Description	Service	Type
	The CMS run history is stored in the WBM DB schemas in OMDS. This data is later accessed by the L1CMS to give feedback to its users	Monitoring	JDBC/OCCI

C Size and Complexity of the Level-1 and its Control and Monitoring System

Table 3 enumerates different figures of merit about the size and complexity of the L1 and L1CMS systems. From left to right the Table shows the subsystem name and its acronym, the number of crates, the number of boards plugged in the VME crates to be configured, the number of different board types, the number of processes meant to configure, monitor and test the given subsystem, and the number of source line of codes (excluding blanks and comments).

Table 3: Several figures of merit of the L1 and L1CMS size and complexity.

Subsystem	Acronym	Crates	Boards	Board Types	Computers	Proceses	Lines of Code [x1000]
Level-1 Page 1	L1 Page	-	-	-	1	1	6.4
Trigger Function Manager	TRG FM	-	-	-	1	2	2.7
Trigger Supervisor Framework	TS	-	-	-	-	-	35.2
Central	Central	-	-	-	1	9	1.2
Global Trigger	GT	1	10	6	1	8	144.7
Global Muon Trigger	GMT	1	4	2	1	1	19.9
Global Calorimeter Trigger	GCT	7	74	4	3	9	70.4
Cathode Strip Chamber Track Finder	CSCTF	1	14	3	1	9	104.7
Drift Tube Local Trigger	DTLT	260	1780	4	5	18	35.6
Drift Tube Track Finder	DTTF	7	107	8	3	20	39.1
Resistive Plate Chamber Trigger	RPC TRG	109	1511	7	5	52	216.2
Regional Calorimeter Trigger	RCT	18	288	4	10	27	50.0
Trigger, Timing and control Machine Interface	TTC	1	4	3	1	10	7.9
Trigger, Timing and Control	TTC	9	40	3	9	38	62.9
TOTAL	**-**	**425**	**4484**	**38**	**42**	**204**	**797.1**