Vienna University of Technology

Institute of Software Technology and Interactive Systems

Favoritenstrasse 9-11/188, 1040 Vienna

Suitability Study of the CMS Trigger Supervisor Control Panel Infrastructure: The Global Trigger Case

Diplomarbeit

zur Erlangung des akademischen Grades

„Diplom-Ingenieur“

Student:	Alexander Winkler
	Student ID: 0126932
TU Vienna:	Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gerald Futschek
CERN:	DDipl.-Ing. Ildefons Magrans (HEPHY Vienna)
	Dipl.-Ing.Mag. Marc Magrans (CERN)

Abstract

The software level expert tools for the Level 1 Trigger at the CMS Experiment, which is located at CERN in the LHC tunnel, have been previously developed heterogeneously as standalone tools. A homogenous design with solely one web access point will shorten the operators' learning curve, ease the maintenance of the overall Level 1 Trigger software infrastructure and enhance sharing of experience and source code. The hypothesis to be proven says that the Trigger Supervisor Control Panel infrastructure provided by the Trigger Supervisor Framework is suitable for the homogenization of the Level 1 Trigger online expert tools. This document describes the development and integration of the most complex standalone expert tool as the first Control Panel within the Trigger Supervisor, namely the one for the Global Trigger.

Abstrakt

Die Software-Level Expertensysteme für den Level 1 Trigger des CMS Experiments, welches sich auf dem CERN im LHC Tunnel befindet, wurden früher mit heterogenen Technologien als selbstständig lauffähige Applikationen entwickelt. Ein homogenes Design mit einer einzigen zugänglichen Schnittstelle verkürzt die Lernkurve des Operators, vereinfacht die Wartung der Level 1 Trigger Software-Infrastruktur und fördert den Zugriff als auch die Wiederverwertung von vorhandenem Source Code. Die zu beweisende Hypothese besagt, dass die Trigger Supervisor Control Panel Infrastruktur, die vom Trigger Supervisor zur Verfügung gestellt wird, für die Homogenisierung der Level 1 Trigger Online Expertensysteme geeignet ist. Dieses Dokument beschreibt die Entwicklung und die Integration des kompliziertesten selbstständig laufenden Systems als das erste Control Panel innerhalb des Trigger Supervisors, nämlich das Global Trigger Control Panel.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Wien, 14.2.2008

Alexander Winkler

Index

1 Introduction

The Level 1 Trigger decides about the importance and therefore about the recording of each bunch crossing event at the CMS Experiment at CERN. The software level expert tools for the Level 1 Trigger have been previously developed as standalone tools implemented with a heterogeneous spectrum of technologies.

The goal of the master thesis is to prove the Trigger Supervisor Control Panel concept drafted within the Trigger Supervisor Framework. This proof happens by the development and integration of the most complex standalone expert tool as the first Control Panel within the Trigger Supervisor. The one for the Global Trigger is called Global Trigger Control Panel.

The development comprises the evaluation of the Global Trigger Control Panel requirements, the design of the classes and the implementation of the project. Additionally bugs within the AjaXell library are reported to the AjaXell developers. The integration comprises the modification of the Trigger Supervisor classes in order to incorporate the Global Trigger Control Panel into the Trigger Supervisor Generic GUI. Afterwards a test phase is necessary in order to validate the proper functionality.

The proof will open doors for the migration of existing standalone tools to Control Panels and therefore contribute to the harmonization of the Level 1 Trigger expert tools. The harmonization will provide the following benefits:

1. Learning curve of the operators will become shorter.
2. The maintenance of the overall Level 1 Trigger software infrastructure will be simplified by eliminating standalone tools which were previously implemented amongst others by using Tcl/Tk, Java, Qt and Command line.
3. Sharing of code and experience between subsystems will be enhanced.

The greatest challenges to be met for this master project are the acquirement of a new technological skill set consisting of C++ and AJAX and the close interaction with physicists requiring a basic understanding of the physics processes being run within the scope of the project.

This document is subdivided as follows. Section 2 gives an introduction to the given scientific environment. It consists of the Large Hadron Collider, the Compact Muon Solenoid experiment and the Level 1 Trigger. The Level 1 Trigger includes the Global Trigger, in which the development and the integration of the Global Trigger Control Panel took place. Section 3 provides the functional and the non-functional requirements to the Global Trigger Control Panel. Section 4 explains the class architecture of the Global Trigger Control Panel and the interdependencies between the classes. Section 5 provides information about the testing procedure. This was used in order to verify the usability and the correct data flow between the Control Panel and the hardware. Section 6 gives an insight into performance measurement and optimization steps. These steps were applied to the Global Trigger Control Panel in order to improve its response time. Section 7 depicts the necessary classes used for the integration of the Control Panel into the Trigger Supervisor Framework. Section 8 summarizes the achievements and outcomes of the master thesis project.

Beside section 2 which describes the broader context of the Global Trigger Control Panel and beside section 7 which depicts a software architecture already existing before the start of my project the products of the other chapters were elaborated by me.

2 Introduction to the Scientific Environment

Chapter 2 gives a brief introduction to the scientific environment. First it describes the Large Hadron Collider (LHC) including its physics experiments and its goals. Second the master project was performed within the scope of the Compact Muon Solenoid (CMS) Experiment. Therefore the CMS Experiment is explained in more detail. Third the Level 1 Trigger (L1T) including its Global Trigger (GT) is introduced. The Trigger Supervisor (TS) Framework is responsible for monitoring and operating the L1T. Also the Run Control Management System (RCMS) is described as part of the distributed control system.

2.1 Large Hadron Collider (LHC) at CERN

2.1.1 Overview

The LHC [1] is being built in order to find new physics at high energies. It is a particle accelerator located at CERN [2] which will be switched on in 2008. It has a circumference of 27 kilometers, crossing the Swiss and French borders and is buried at around 50 to 175 m depth. The tunnel is the same as the one used for the Large Electron Positron (LEP) Collider during the 90's. LHC will collide two counter rotating beams (see Figure 1) of heavy ions or protons. The beams are moved inside a continuous vacuum and kept on track by superconducting magnets. A huge cryogenics system is cooling these magnets down to about 300 degrees below room temperature. Inside the four main LHC experiments proton-proton collisions will occur at energies of 7 TeV per beam.

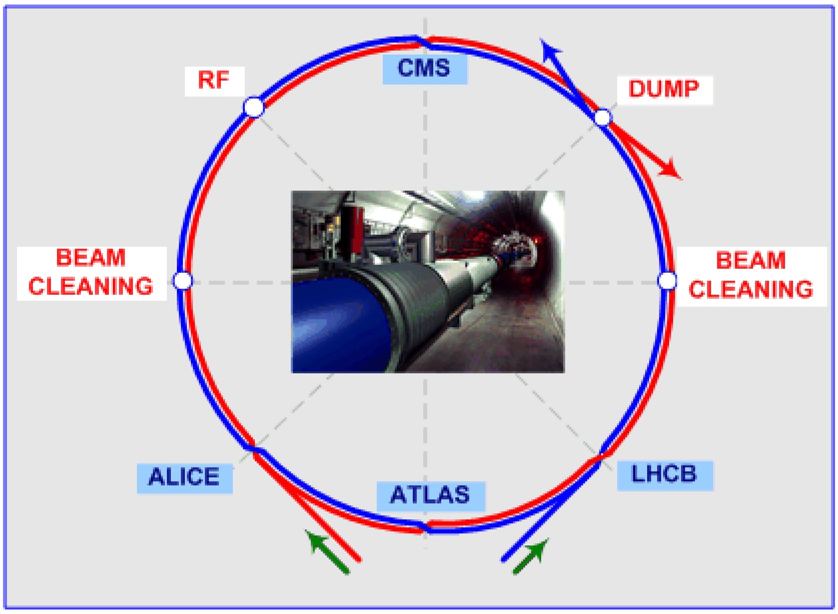

Figure 1: Two beams within the LHC collider [1].

The LHC will create a post big bang similar environment for probing deeper into matter than ever before and thereby answering amongst others the following physical questions:

- *Higgs:* The Standard Model includes a yet undiscovered particle, the Higgs boson, which is necessary to give mass to certain particles. The Higgs-Boson is assumed to be neutral and to have spin-0. Its mass is not predicted by the model. Depending on whether the particles interact heavily or lightly with the Higgs-Field their mass is big or small. If existing, the Higgs-Boson should be found at LHC.
- *SUSY:* For each particle there exists a supersymmetric partner. Superpartners have a spin difference of ½ with respect to the Standard Model particles. Supersymmetric particles (sparticles) must be heavy since none of them has been discovered so far. If existing, the supersymmetric partners should be found at LHC.
- *Dark Matter:* In astrophysical context the question arises if Dark Matter exists.
- *Dark Energy:* This energy accelerates the expansion of the universe and accounts for ¾ of the universe's mass.

2.1.2 Experiments at LHC

A Toroidal LHC ApparatuS (ATLAS)

The ATLAS experiment [3] is one of the main LHC experiments and one of two general-purpose experiments. Its goal is to evaluate high-energy proton-proton collisions in order to find the Higgs-Boson. Also SUSY is hoped to be proven. The detector is built onion-layered in order to examine different particle signatures.

Large Hadron Collider beauty (LHCb)

LHCb [4] is a single-arm spectrometer. It has a forward coverage from approximately 10 mrad to 300 mrad in the bending plane. Its goal is to measure CP violations with B-mesons.

A Large Ion Collider Experiment (ALICE)

ALICE [5] is a heavy ion detector. It will measure the flavor content and phase-space distribution event by event for a large number of particles whose masses and momenta are of the order of the typical energy scale involved. This experiment will cope with the highest particle multiplicities anticipated.

Compact Muon Solenoid (CMS)

CMS is the second of two general-purpose experiments for the LHC [6]. It will examine proton-proton collisions. Special compared to ATLAS is the scintillating crystal calorimeter for high energy protons (see Section 2.2 for more details). CMS and ATLAS are designed to prove each other’s results.

2.2 CMS Experiment

2.2.1 Overview

The CMS Experiment is a general-purpose experiment. CMS’ main feature is a strong solenoidal magnetic field. CMS consists of a crystal electromagnetic calorimeter, an inner tracker with an embedded pixel detector, a copper-scintillator hadron calorimeter and a muon system made up of special trigger chambers and tracking chambers. Figure 2 shows the composition of the CMS Experiment, which is described in sections 2.2.2 - 2.2.5.

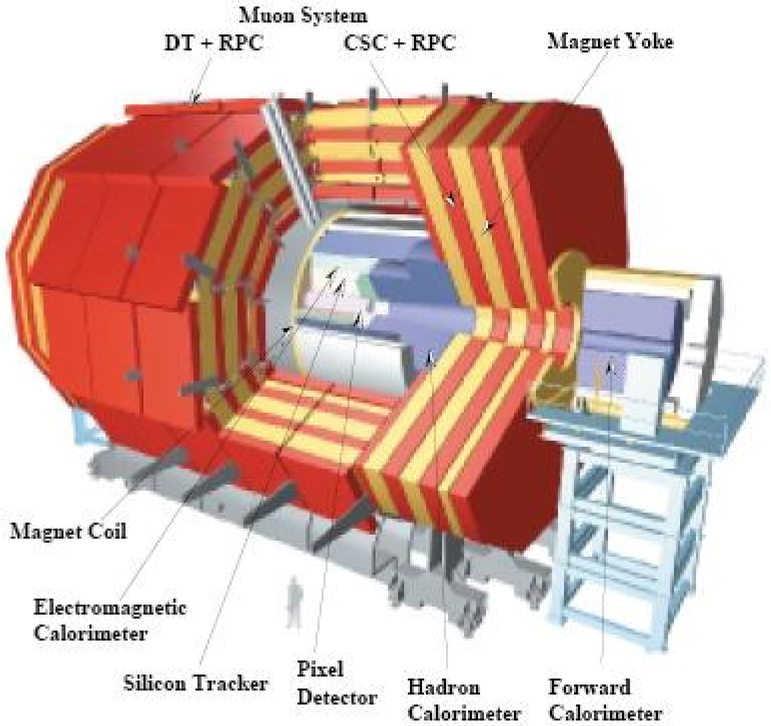

Figure 2: CMS Experiment composition [16].

2.2.2 Tracker

The Tracker reveals the positions at which unstable particles decay and measure the momenta of charged particles. The detectors are arranged in following way.

- Pixel detectors [7] are placed closest to the interaction vertex where the particle flux is the highest (about 10^7/s at r = 10 cm)
- Silicon microstrip detectors [8] are used in the intermediate region (20 < r < 55cm), where the particle flux is low enough.
- Larger-pitch silicon microstrips are used in the outermost region (r > 55cm) of the inner tracker, where the particle flux has dropped significantly.

The silicon microstrip detectors in the barrel part are placed at *r* between 20 and 110 cm. The forward region has two pixel and nine microstrip layers in each of the two endcaps. The barrel tracker region is divided into two parts; a Tracker Inner Barrel (TIB) and a Tracker Outer Barrel (TOB). The inner barrel is shorter than the outer barrel in order to avoid excessively shallow track crossing angles. Additionally there are three inner disks in the transition region between the barrel and endcap parts on each side of the inner barrel. The total area of the pixel detector is about 1 m^2, while that of the silicon strip detector is 200 m^2 providing coverage up to a rapidity range of

$|\eta| < 2.4$. The inner tracker comprises 9.6 million silicon strips and 66 million pixels which will be described briefly in the subsequent sections.

Strip Tracker

The TIB is made of four layers and covers the effective nuclear electron charge of up to $|z| < 65$ cm using silicon sensors with a thickness of 320 µm and a strip pitch varying from 80 to 120 µm. The TOB has six layers with a half length of $|z| < 110$ cm. Small radiation levels provide a good signal to noise (S/N) ratio for longer strip length and wider pitch. The silicon sensors are 500 µm thick and the strip pitch varies from 120 to 180 µm.

The endcaps consist of the Tracker Inner Disk (TID) and Tracker End Cap (TEC). Each TID consists of three disks that fill the gap between TIB and TEC. Each TEC consists of nine disks which extend into the region 120 cm < $|z|$ < 280 cm.

The entire silicon strip detector (see Figure 3) comprises almost 15.400 modules housed within a temperature controlled outer support tube and mounted on carbon-fiber structures.

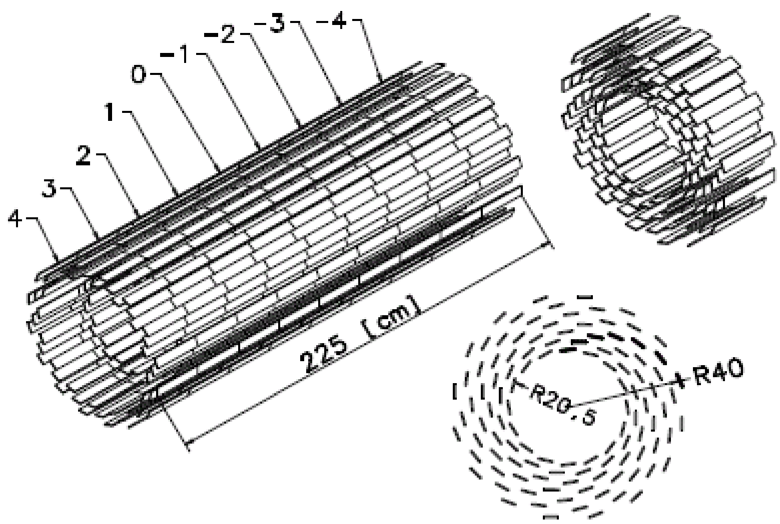

Figure 3: Layout of the barrel silicon detector [6].

Pixel Tracker

The pixel detector comprises three barrel layers with two endcap disks on each side (see Figure 4). Each pixel module has the size 100 x 150 µm^2 (in (r, Φ) and on z-coordinate). 768 pixel modules are arranged into half-ladders of four identical modules each form a barrel. A turbine-like geometry is used for assembling the endcap disks.

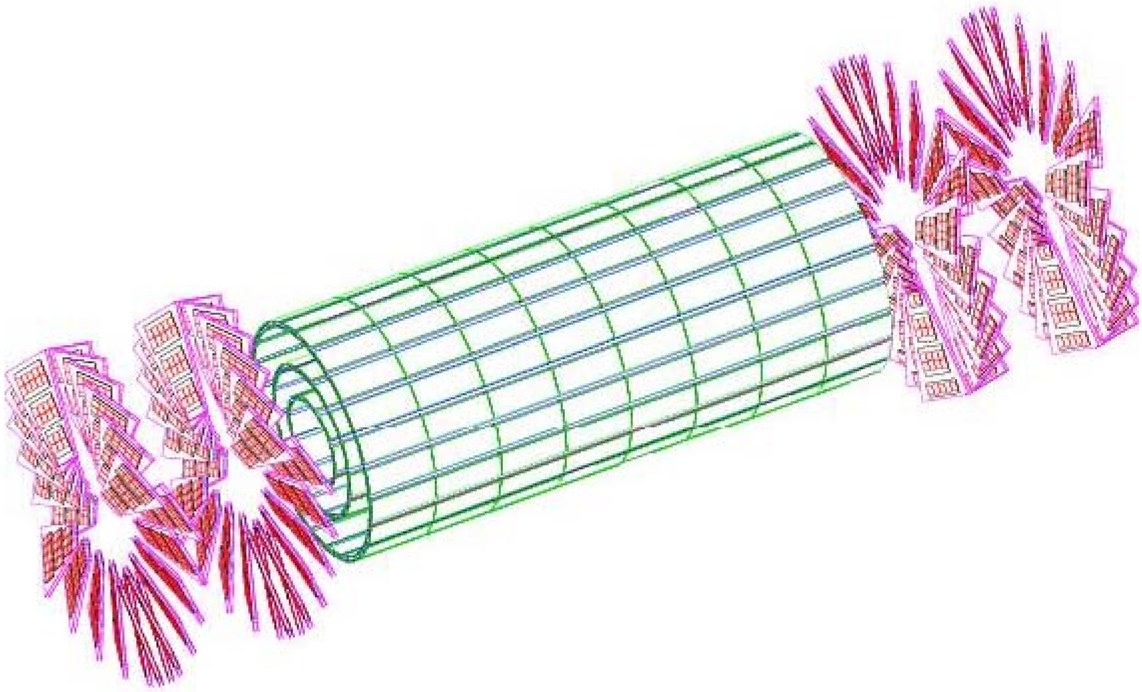

Figure 4: Layout of the CMS Pixel Tracker with its end caps [7].

2.2.3 Electromagnetic Calorimeter (ECAL)

The ECAL (see Figure 5) is a high precision scintillating crystal calorimeter. The ECAL will contribute massively to the search for the Higgs. It has to work within an extreme environment, whereby every 25 ns an average of 20 events with some 1000 charged tracks will be generated. Big effort has been put into developing appropriate crystals, electronics, photo detectors and software. The ECAL provides the best performance since most of the energy from photons or electrons is available within the homogeneous crystal volume of the calorimeter. The crystals chosen are lead tungstate ($PbWO_4$) crystals. $PbWO_4$ has a small Moliere radius and a short radiation length. Therefore it is a fast scintillator which can be produced from existing raw materials.

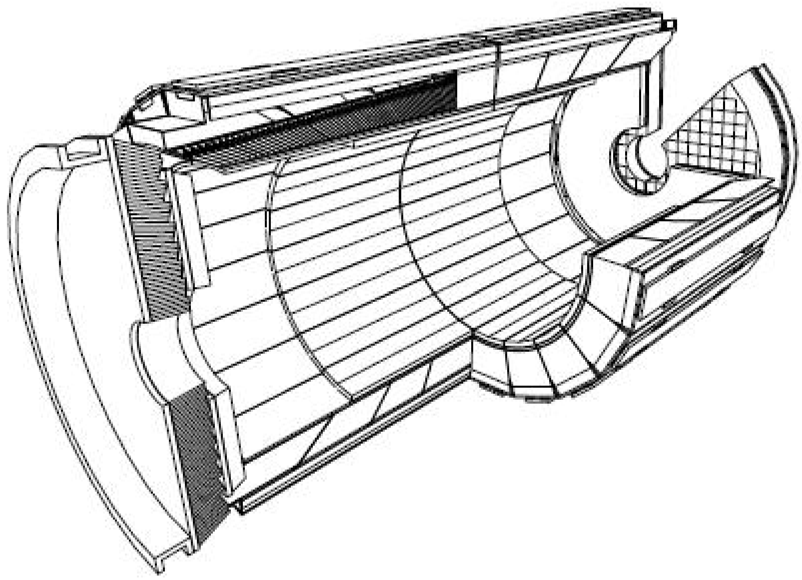

Figure 5: The Electromagnetic Calorimeter of the CMS experiment [9].

2.2.4 Hadron Calorimeter (HCAL)

The HCAL [10] will measure gluon, quark and neutrino directions and energies. Therefore it will measure the direction and energy of particle jets of the missing transverse energy flow. The identification of missing energy will provide a crucial signature for new phenomena and particles. These might be relevant for the search of SUSY partners of quarks and gluons. The HCAL will also contribute to the identification of photons, electrons and muons in conjunction with the muon systems and the ECAL.

2.2.5 Muon Chambers

The muon detector [11] has to fulfill three tasks: muon identification, momentum measurement and triggering. It is located behind the calorimeters and the coil and consists of four muon stations (see Figure 6) which are at mean radii of 4.0, 4.9, 5.9 and 7 m from the beam axis and are interleaved with the iron return yoke plates. In both the barrel and the forward regions, a space of 40 cm is available for four muon stations. The first 5 cm are used as air gap. The remaining space is used for the chambers, which will be described in the subsequent sections.

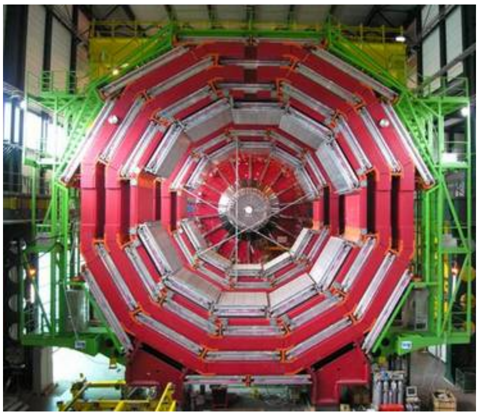

Figure 6: Lateral cut of the CMS Experiment showing the four Layers of the Muon Chambers/Stations [7].

Drift Tube Chamber (DTC)

The DTCs (see Figure 7) are used where the neutron induced background is small and the muon rate as well as the magnetic field in the chambers is low. This is true for the barrel region ($|\eta| < 1.2$). Each of the five wheels of the barrel detector is separated into 12 sectors. A Punch-through is induced by pions, which pass through the calorimeters [13]. Chambers in different stations are constructed so that high-p_T (punch through) muons produced near a sector boundary will have to cross at least three out of four stations. The single point resolution is about 200 μm. Each station will give a muon vector in space with approximately 1 mrad in Φ direction and a precision better than 100 μm in position.

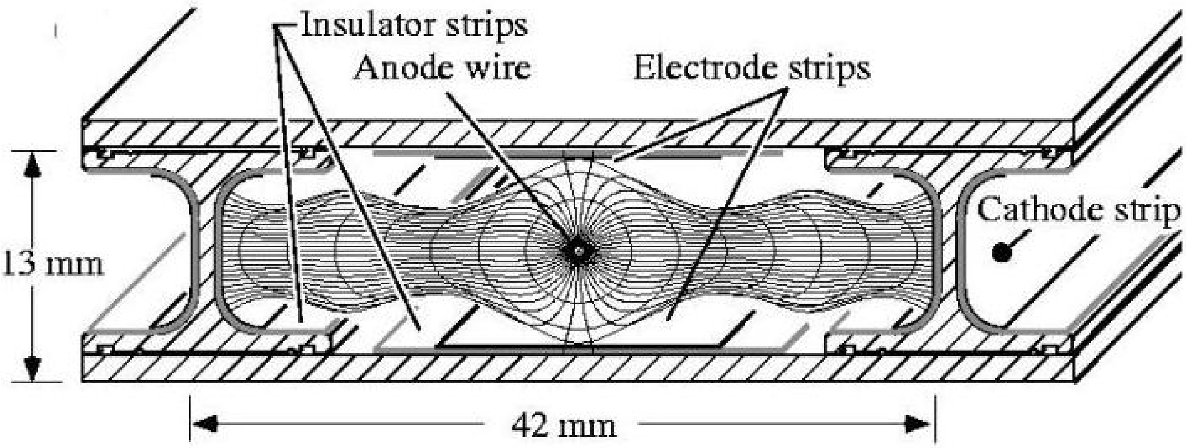

Figure 7: Lateral cut of the Drift Tube Chamber [12].

Cathode Strip Chamber (CSC)

CSCs are used in the region up to $|\eta| < 2.4$. The Muon Endcap (ME) system consists of two endcaps containing 468 CSCs. Each ring of a muon station contains 36 chambers. The innermost ring of the second through fourth disk contains 18 chambers. A charged particle which traverses each plane of a chamber causes a gas ionization and subsequent electron avalanche whereby producing an image charge on a group of cathode strips and a charge on the anode wire. The cathode strips induce the charge distribution whereby the centre-of-gravity can be determined and the position precisely measured.

Resistive Plate Chambers (RPC)

RPCs are used as well in the barrel as in the endcap regions. Avalanche mode ensures good operation at high rates (up to 10 kHz/cm^2). The RPCs are assembled out of two parallel resistive plastic plates which are separated by a few millimeters gas gap. They offer a fast response and a good time resolution. The position resolution is worse than the one of DTCs and CSCs however, RPCs can measure the time of occurrence of a bunch crossing with a precision better than 2 ns. The arrangement of DTC and RPC is shown in Figure 8.

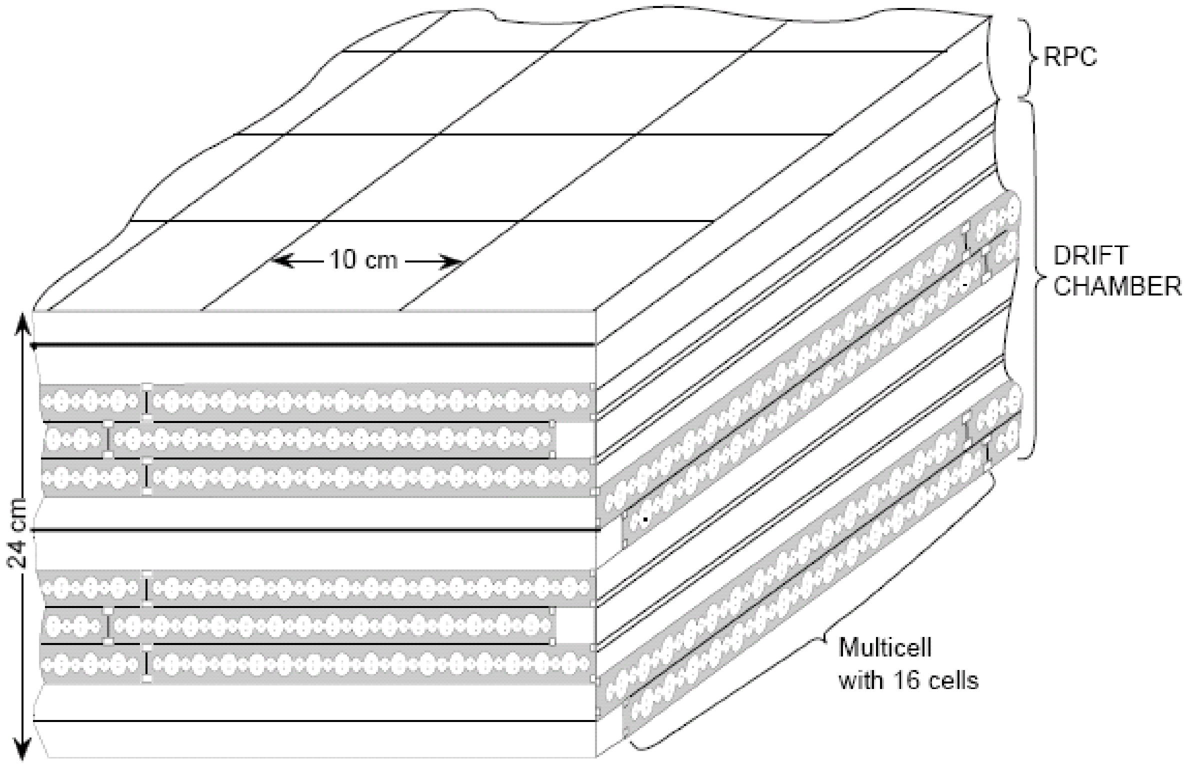

Figure 8: Arrangement of one muon station [11].

2.3 CMS Trigger

A highly selective online data selection process and a sophisticated high bandwidth data acquisition system are needed to store the rare processes which may occur at high collision energies and rates. The accelerator will produce heavy-ion and proton-proton collisions at high interaction rates. The crossing interval for proton bunches within the beam is 25 ns. The average interaction rate is 40 MHz with a mean event size of 1 MB (i.e. 40 TB/s of data). The amount of data generated can neither be stored nor transported. Therefore a reduction mechanism has been introduced, the so-called Trigger System which is the beginning of the physics event selection process. The rate is sequentially reduced in two stages called Level 1 Trigger (L1T) (see section 2.4) and High Level Trigger (HLT) [14]. The first is a custom hardware system, and the second is a computer farm running a distributed software application. The rate reduction for the combined L1T and HLT will be at least a factor of 10^6. The maximum allocated bandwidth to the L1T is 100 kHz and will be reduced to 50 kHz at the startup of LHC due to operational reasons. In the second stage, the HLT will reduce the rate to 100 Hz (i.e. 100MB/s of final data throughput). This document describes solely the scope of the L1T.

2.4 Level 1 Trigger (L1T)

2.4.1 Overview

Due to limited disk space and due to the high frequency of bunch crossing events a mechanism is needed to reduce the rate of stored and processed events for the CMS experiment. The L1T [15] has been created as a custom built hardware system which decides whether to accept or reject events within a few microseconds after the collision. It consists of the calorimeter trigger systems and the muon trigger systems which generate trigger primitives and send them to the Global Trigger (GT) [16] (see Figure 9).

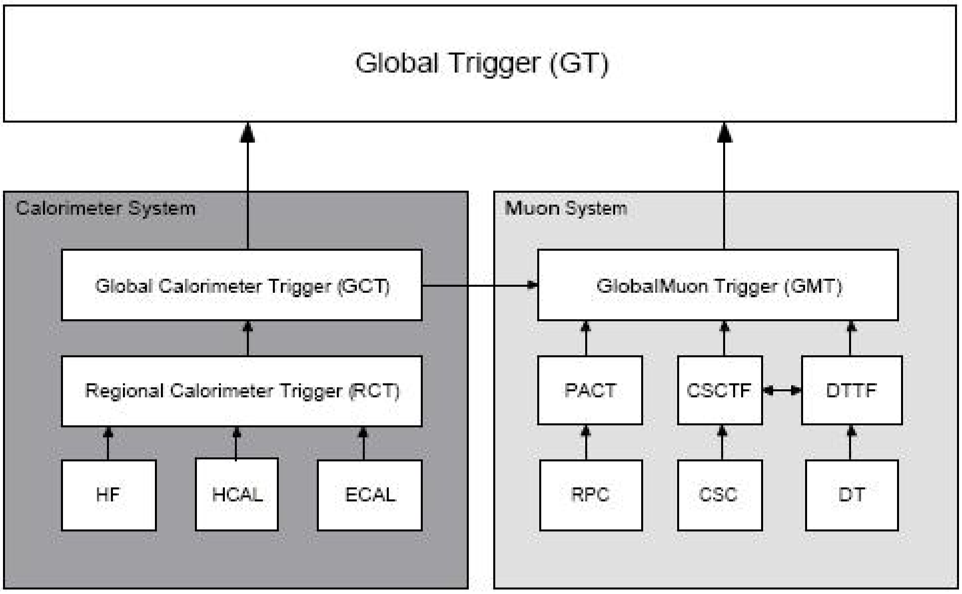

Figure 9: Composition of the L1 Trigger [17].

2.4.2 Calorimeter System

There exist three calorimeters detectors: Forward Hadronic Calorimeter (HF), Central Hadronic Calorimeter (HCAL) and Electromagnetic Calorimeter (ECAL). These detectors send information to the Regional Calorimeter Trigger (RCT) in order to identify the best photon, electron or hadron jet candidates. Afterwards, the candidates are sent to the Global Calorimeter Trigger (GCT) and sorted by quality, correlation, and p_T. The best four in terms of p_T and quality are sent to the GT.

2.4.3 Muon System

There exist three muon detectors having its own trigger logic: Resistive Plate Chamber (RPC), Cathode Strip Chamber (CSC) and Drift Tube (DT). These detectors send information to their corresponding trigger track finders - Cathode Strip Chamber Track Finder (CSCTF), Drift Tube Track Finder (DTTF) and Pattern Comparator Trigger (PACT) in order to identify the muon tracks. The Global Muon Trigger (GMT) validates the muon charge sign, converts the track parameters from all subsystems to the same scales, and finally, attempts to correlate the tracks of the different subsystems in order to improve the quality of the best candidates. It sends the best in terms of p_T and quality four muons to the GT. Figure 10 shows the position of the three detectors. CSCs are allocated in the forward region. DTs are allocated in the barrel region outside of the magnet coil.

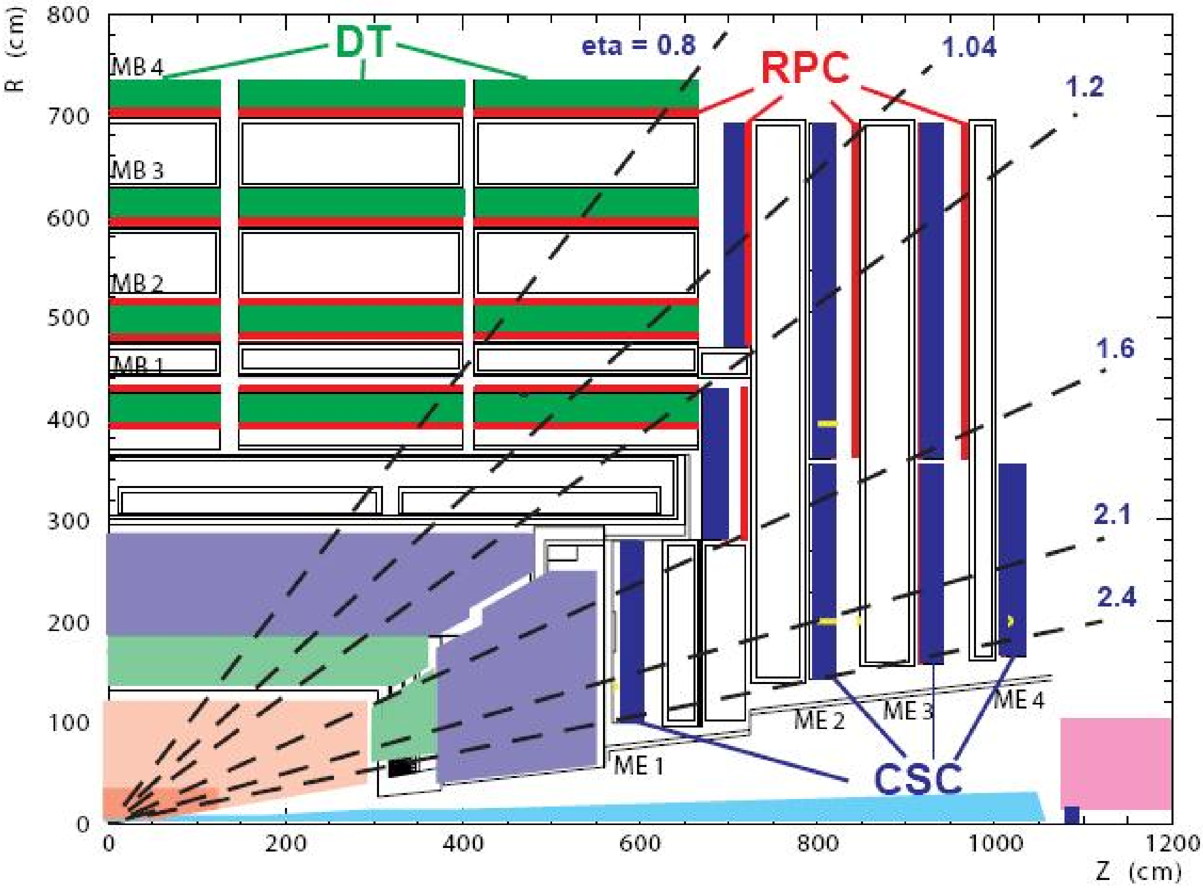

Figure 10: Longitudinal cut of the CMS Muon System [7].

2.4.4 Global Trigger

The GT is the final stage of the L1T. Its decisions are based on the trigger objects provided by the GCT and the GMT. Trigger objects can be jets, candidate electrons or muons. The determination happens in three logical steps. First the muon and calorimeter system generate local trigger information, which is called Trigger Primitives. Second and third are the calculations of global and regional subdetector specific quantities. These steps are performed as one step for the RPCs since they don't diversify between regional and global trigger systems. The global quantities are forwarded to the GT.

The ultimate goal of the GT is to generate - dependent on the Trigger Primitives - L1 Accept/Reject (L1A) signals and send them to the Timing, Trigger and Control (TTC) System [18], which itself delivers it to the different subdetector front-end controllers.. The GT must decide about events every 25 ns without any interruption. The main part of the Global Trigger Processor logic is the trigger algorithms. An algorithm is the combination of trigger objects fulfilling threshold, space and quality requirements. Up to 128 algorithms can be programmed for the physics run. All algorithms are processed concurrently and results are delivered as a string of 128 bits. The final decision regarding the L1A is taken by applying a final OR on the 128 algorithm bits.

The GT is a complex electronic system consisting of several VERSAmodule Eurocard Bus (VME) modules mounted in a VME9U crate (see Figure 11) together with the central Trigger Control System (TCS) and the GMT. The modules are described briefly in the subsequent sections.

.

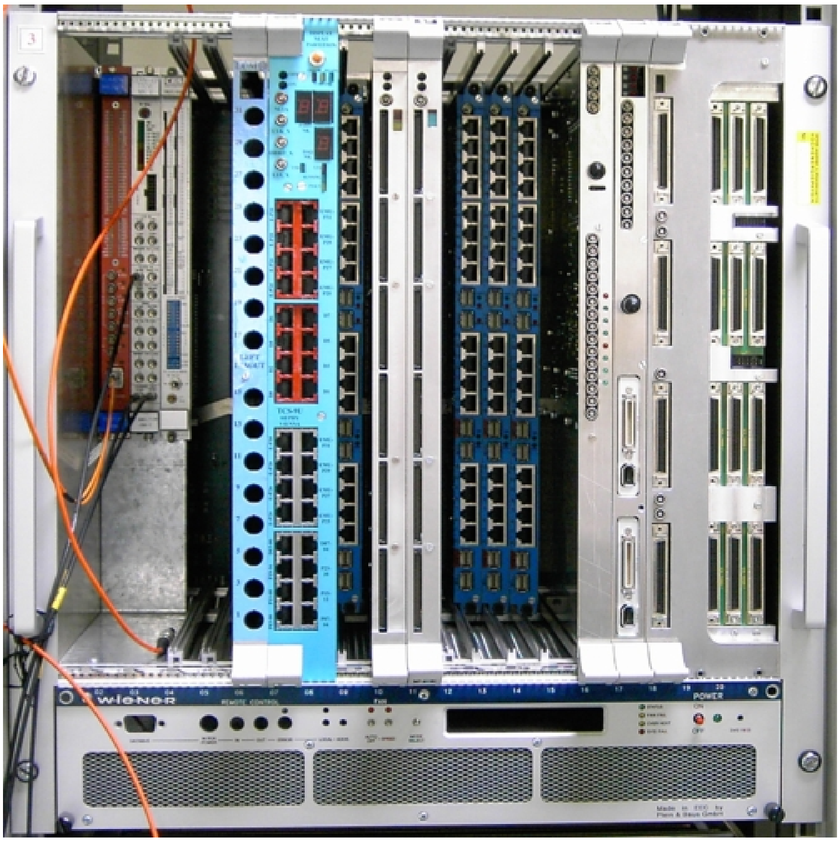

Figure 11: The GT crate with its boards [29].

Trigger Control System Board (TCS)

The TCS board [19] controls the delivery of L1A. It also generates Bunch Crossing Zero and Level 1 Reset commands and controls the delivery of calibration and test triggers. Bunch Crossing Zero commands reset the counters in order to begin a new

LHC orbit. The Level 1 Reset command resets the hardware to the original status. The board receives the status of the L1A and distributes information to the subsystems through the TTC network.

The Trigger Control Logic divides the CMS readout system into 8 TTC/DAQ-Partitions (for more information see section 2.4.5). The Global Trigger Processor generates maximum 1 Final-OR for each TTC/DAQ-Partition.

Pipeline Synchronizing Buffer Boards (PSB)

The PSB boards monitor all bits for every bunch crossing and synchronize the fast commands generated by the TCS to the GT clock. The PSB input modules receive 12 input channels for synchronizing all input channels to each other and to the LHC orbit. Three PSB boards are used for the GMT, two are used to accept fast control signals for the TCS and three are used for the GT logic. All input channels but the muons are sent to the Global Trigger Logic (GTL) [20] module via the backplane. This happens with a programmable delay which keeps the overall L1 latency as small as possible.

Final Decision Logic Board (FDL)

The FDL board computes the L1A signal for each of the 8 TTC/DAQ-Partitions. The L1A is computed using an OR function that combines 128 algorithms bits from the GTL board and 64 technical trigger (triggers containing Beam Scintillator Counters) bits from a dedicated PSB board. These signals are sent to the TCS board, if the Trigger Throttling Rules accept the signals they will be forwarded to the front-end electronics in order to read-out the corresponding event. The Trigger Throttling System (TTS) [21] guarantees that the trigger logic is not overloading the electronic devices within the data flow from the front-end to the storage media. Technical Triggers and Trigger Algorithms are defined as “slices”, which can be accessed by a rate counter.

Global Trigger Logic Board (GTL)

The GTL receives the best four muons from the GMT. The GTL board combines the data from GMT and GCT and calculates up to 128 trigger algorithms in parallel. It delivers the algorithm bits to the FDL module in order to be prescaled. Thereby for L1A rates, that are too high, a prescale factor can be applied for each slice.

Timing Board (TIM)

The TIM [22] is responsible for the distribution of the clock and fast control commands to all the boards of the GT crate.

2.4.5 Control Mechanism

The Run Control and Monitoring System (RCMS) [23] is the control system of the CMS experiment. It is the collection of hardware and software components responsible for controlling and monitoring the CMS experiment during data taking.

The RCMS subdivides the Data Acquisition (DAQ) into up to eight TTC/DAQ-Partitions. These partitions can be accessed independently and are used for configuring and operating the trigger and readout electronics. Each TTC/DAQ-Partition can take care of a subset of the 32 detector groups (i.e. Timing, Trigger and Control System (TTC) Partitions).

Each TTC/DAQ-Partition is controlled by a Session Manager (SMR) of the RCMS. The SMR controls the user access rights and distributes the fast commands to the lower levels. Every subsystem has a corresponding Function Manager (FM). Per TTC/DAQ-Partition one FM is instantiated. It receives requests from the appropriate SMR and provides the Application Program Interface (API) to the subsystem application. This allows transforming requests and forwarding them to the subsystems. Figure 12 shows the CMS control architecture.

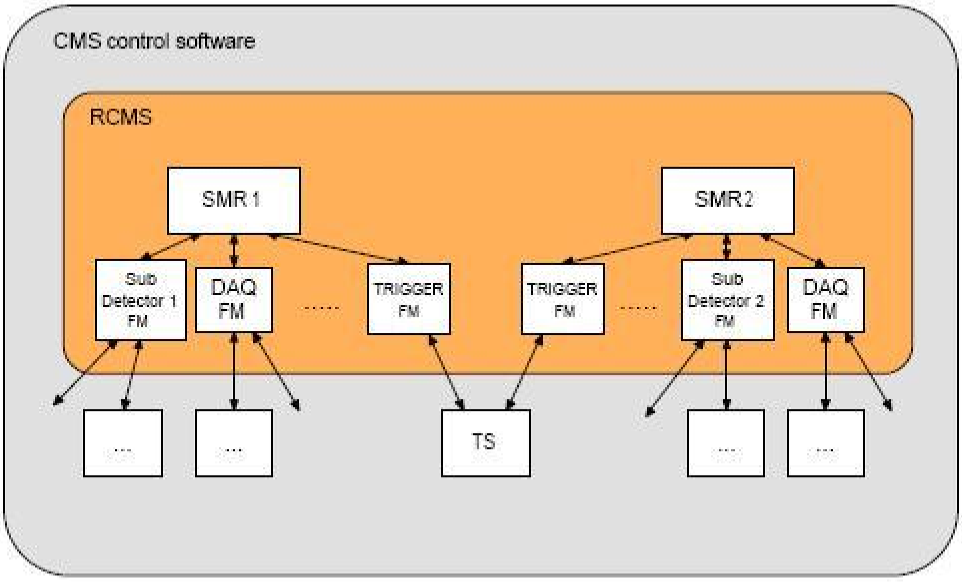

Figure 12: RCMS as an element of the CMS Control Software containing SMR, Subsystems and FMs. The TS is the control node of the L1T in the hierarchical control structure of the CMS experiment [17].

2.4.6 Trigger Supervisor (TS)

The application for the L1T control is the TS System. The TS is a distributed software application designed to set up, monitor and test the trigger components and to manage their information exchange and their interplay with the RCMS [24]. Figure 13 shows the architecture of the TS. The TS System has been built using the TS Framework [25], a software framework based on the Cross-Platform DAQ Framework middleware (XDAQ) [26]. The TS is also a distributed application that controls all the nodes of the L1T. All the TS nodes (in this case called TS Subsystems) are XDAQ applications referred to as Cells. One of them is the GT Cell. The top element of this hierarchical structure is the Central Cell. This can be accessed by several RC sessions concurrently and propagate information to the subsystem cells. The subsystem cells have to be implemented in order to operate the corresponding hardware.

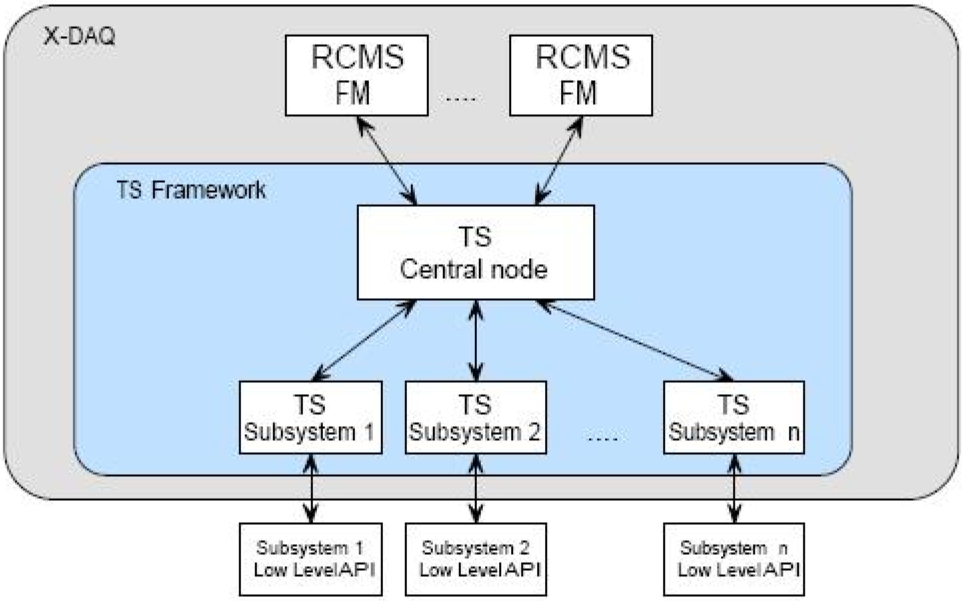

Figure 13: Architecture of the TS System [17].

Currently, there are two interfaces to communicate with the TS Cells (see Figure 14). A machine-to-machine Simple Object Access Protocol (SOAP) [27] interface to communicate between Cells and between the Central Cell and RCMS, and a human-to-machine Common Gateway Interface (CGI) [28] to control the Cells from a web-browser.

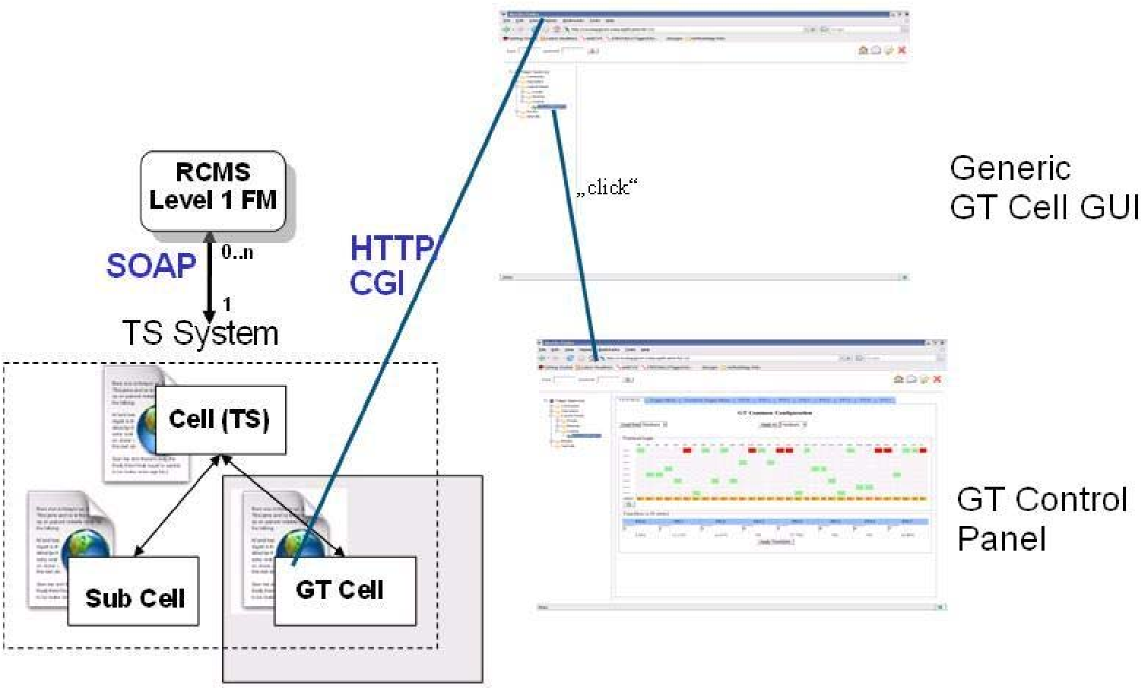

Figure 14: The two available interfaces to access the GT Cell: HTTP and SOAP.

3 Global Trigger Control Panel Requirements

Chapter 3 describes the functional (see section 3.1) and non-functional (see section 3.2) requirements for the GT Control Panel. In order to create the requirements for the GT Control Panel a triple approach has been used. First, an informal weekly follow up of the development of the Control Panel has been arranged. During these meetings the current version of the Control Panel was presented in front of the group in order to boost the discussion and extract the requirements from the users. Second, some formal and public presentations of the state of the GT Control Panel have been held. And finally the old GUI's based on Java Swing and C++ Qt have been used as an example for the desired look and feel.

3.1 Functional Requirements

3.1.1 Monitoring and Control of the Global Trigger Hardware

The GT Control Panel shall implement the most important functionalities to monitor and control the GT hardware [29]. That includes monitoring of the counters and the Timing, Trigger and Control System (TTC) Subdetectors assigned to the TTC/DAQ-Partitions, setting the time slots, enabling and disabling the TTC Subdetectors for a given TTC/DAQ-Partitions, setting the Final Decision Logic (FDL) board mask, starting a run, stopping a run, starting random triggers, stopping random triggers, changing the frequency and step size for random triggers and resynchronization and resetting of each of the TTC/DAQ Partitions.

3.1.2 Configuration Data Base Population Tool

The GT Control Panel shall allow hardware experts to create configuration entries in the Configuration Database (DB) without the need of any knowledge of the underlying schema of the DB. The Configuration DB contains information about the configuration of the GT. Currently the tool being used to populate the configuration DB is a browsable GUI developed with the use of Java Server Pages (JSP) [30].

3.1.3 Access Control Integration

The GT Control Panel shall support different access control levels. Depending on the user of the Control Panel (i.e. an expert, a shifter or any other person) the panel has to visualize different information and allow different tasks to be performed.

3.1.4 Trigger Menu Generation

The GT Control Panel shall allow the visualization and modification of the Trigger Menu. The Trigger Menu is the high level description of the algorithms that will be used in the Final Decision Logic (FDL) board in order to choose the desired events. For each algorithm it shall be possible to visualize and modify the name, algorithm number, prescale factor, algorithm description and condition properties (i.e. threshold, quality, etc.).

3.2 Non-Functional Requirements

3.2.1 User Friendly Design

The usage of the GT Control Panel has to be intuitive and self explanatory. All the functionalities provided have to be clearly structured. The related information has to be visualized in the best case at one glance.

The Control Panel has to be fast (i.e. more than ~ 5 s response time for any change in the Control Panel appears to the user as a broken application). The startup time of the GT Control Panel has to be short. Loading time and switching time between tabs and screens also has to be fast. The information and values edited in the panel have to be applied immediately to the hardware except for the values which are meant to be applied collectively with other values.

The Control Panel has to be browser independent. At least, it has to support the two most common browsers being Internet Explorer and Mozilla Firefox in any screen resolution.

The inputs which can be performed by the users have to be checked for errors. In case of unacceptable values the GT Control Panel has to provide reasonable error messages.

3.2.2 Desktop-like Look & Feel

The GT Control Panel has to provide Desktop-like look and feel. This means that it has to behave like the desktop applications commonly used under Windows, Linux and Macintosh operating systems. Widgets like tabs, sliders and trees shall facilitate work.

3.2.3 Layered Architecture in the Design

The architecture of the GT Control Panel has to be layered. This architecture will allow the developers of different layers to work independently if the interfaces between layers are clearly defined and stable (see section 4.1 for an explanation of the layered architecture underlying the GT Control Panel).

4 Architecture

Chapter 4 gives an overview about the software layers of the GT software architecture. It also describes all classes which were developed within the scope of the master thesis project for the highest software layer being the GT Control Panel

4.1 Overview

The GT subsystem can be controlled through the GT Control Panel. The GT Control Panel offers means to operate, monitor, and test the GT. It is built upon several software layers (see Figure 15).

The GT Control Panel is the highest level layer within the GT software. The GT Control Panel interfaces with the GT Cell software and with the TS Framework. The GT Cell customizes many services provided by the TS Framework e.g. logging, database access, access control, Control Panel customization, monitoring, etc. These same services are used also by the GT Control Panel.

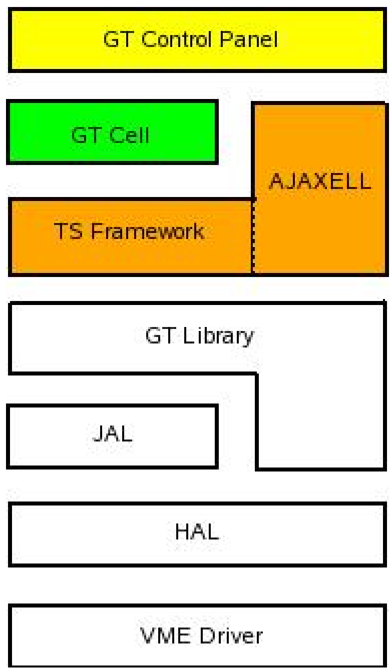

Figure 15: Layered Architecture of the GT Control Panel.

In order to develop a Control Panel a `CellPanel` class descendant must be instantiated. This class allows the integration of the new Control Panel as a plug-in into the TS GUI (see section 7.1 for more information about the generic TS GUI).

Therefore the `CellPanel` descendants allow the TS subsystem developers to customize the Control Panel for their concrete requirements. These classes will encapsulate the look and feel (widgets and related events) of a custom Control Panel for the subsystems. Figure 16 describes the `CellPanel` main dependencies.

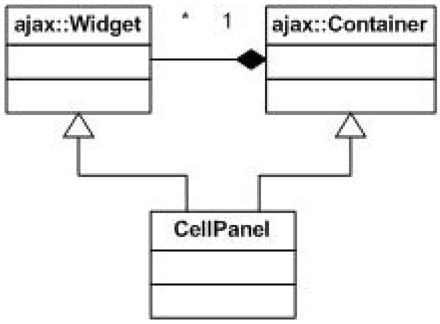

Figure 16: Relationship between CellPanel, ajax::Widget and ajax::Container.

The class `CellPanel` contains one method called `layout()`. This method is used by every inherited class to display the corresponding part of the Control Panel.

The namespaces `gtgui` and `gttoolbox` have been introduced. The first wraps all classes belonging to the GT Control Panel. The latter has been created to encapsulate all calls to the GT Cell layer. This layer is implemented using `CellCommand` and `CellOperation` descendants of the TS Framework. The `gttoolbox` namespace uses routines that wrap the calls to these classes in order to get access to the hardware functionalities.

4.2 Class Structure

An overview of the class structure of the GT Control Panel is shown in Figure 17. All the classes shown in the figure but `MonitorSource` were developed within the scope of the master thesis project. They are descendants of the `CellPanel` class. `MonitorSource` is the container of the Monitorable Items definition of the GT (see [31] for more information about monitoring within the TS Framework).

The classes shown in Figure 17 are described in subsections 4.2.1 - 4.2.14.

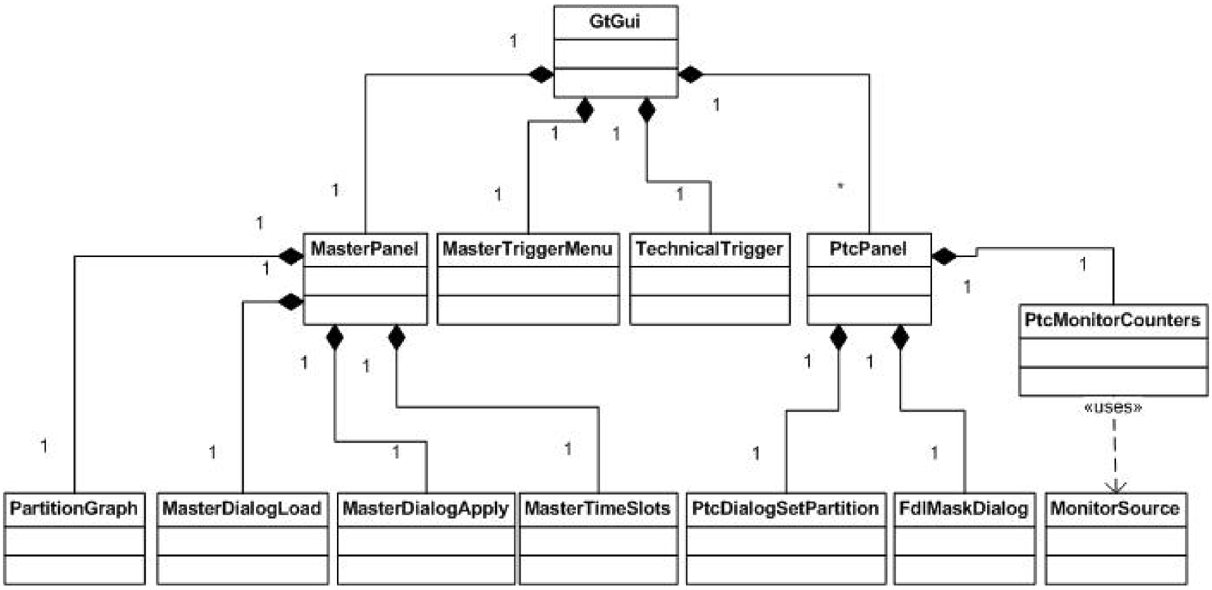

Figure 17: Class Diagram of the GT Control Panel.

4.2.1 GtGui

The class `GtGui` (see Figure 18) is the root class of the GT Control Panel. It contains all widgets and `CellPanel` descendants that correspond to the GT Control Panel.

The visual content comprises a `TabContainer`, which contains eleven tabs. Eight out of the eleven tabs are used to display the eight TTC/DAQ Partitions, one tab is used for the common functionalities of the GT (the "CENTRAL" tab) and, two tabs are used for the Trigger Menu and for the Technical Triggers.
The behavior is encapsulated in the event handler that corresponds to the `OnFocus` event of the tabs in the `TabContainer`. By clicking on the tab the content of the tab is being refreshed.

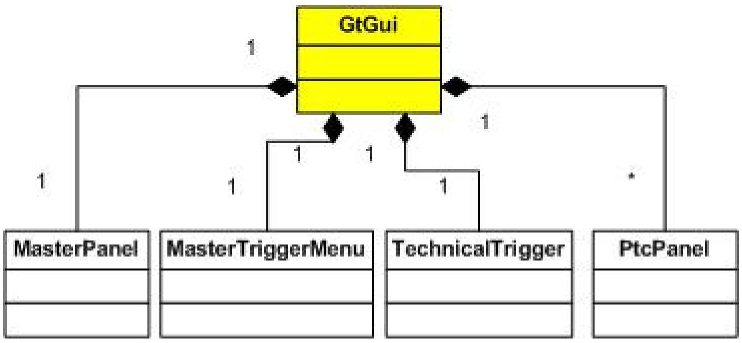

Figure 18: Relationships associated with class GtGui.

4.2.2 Master Panel

The class `MasterPanel` (see Figure 19) is contained by the class `GtGui` and it is used to display and control common configuration items that are not specific to a concrete TTC/DAQ Partition.

The visual content comprises a graph, a table and two buttons. The graph shows the assignment of TTC/DAQ Partitions to subdetectors. The table shows the time slots allocation. Since only one TTC/DAQ-Partition can be operated at one time, time slots need to be assigned. The buttons will be used to apply and load either from/to database or from/to hardware.

The behavior is encapsulated by two events. One event is for refreshing the content of the graph and another for submitting the time slot assignment.

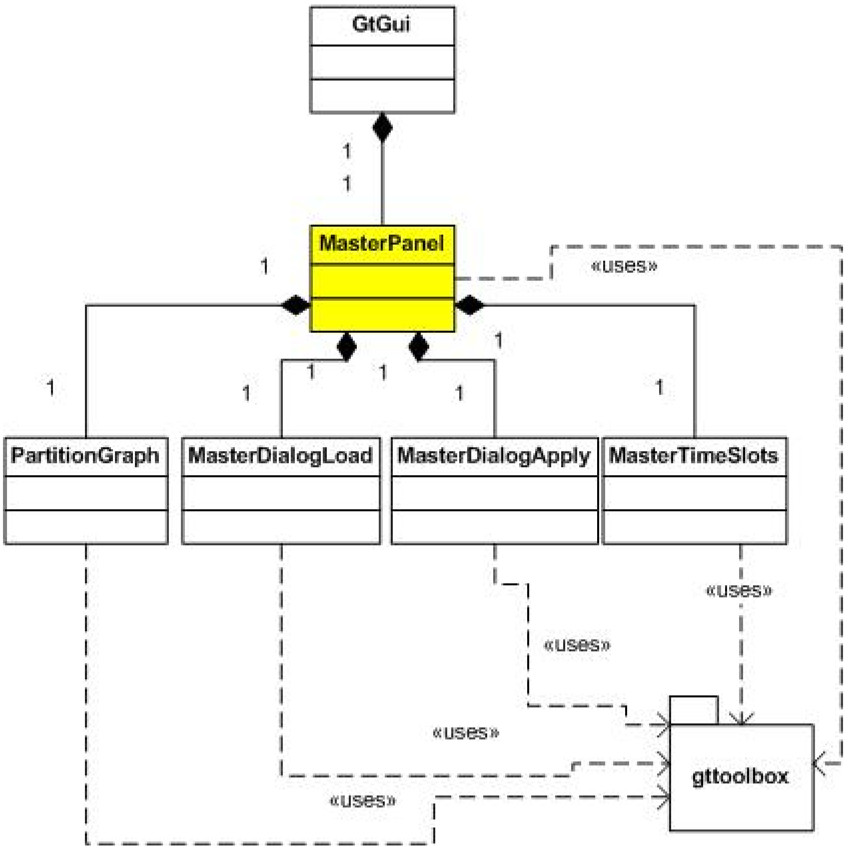

Figure 19: Relationships associated with class MasterPanel.

4.2.3 MasterTriggerMenu

The class `MasterTriggerMenu` (see Figure 20) is contained by the `GtGui` class and is used to visualize and modify the Trigger Menu.

The visual content comprises all (by the time being 128) algorithms including their number, their rate and their prescale factor.

The behavior is encapsulated by the `OnChange` event of the prescale `InputText` .

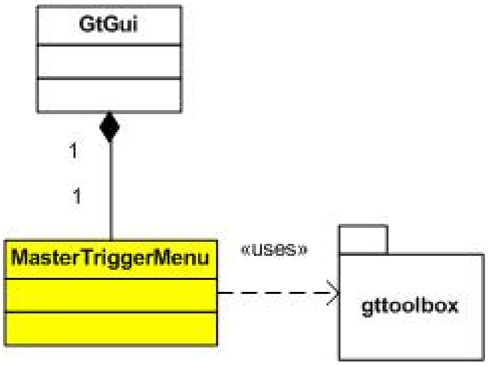

Figure 20: Relationships associated with class MasterTriggerMenu.

4.2.4 TechnicalTrigger

The class `TechnicalTrigger` (see Figure 21) is contained in the `GtGui` class.
The visual content comprises up to 64 Technical Trigger slice information including their number, trigger rate and prescale.

The behavior is encapsulated by the `OnChange` event of the prescale `InputText`.

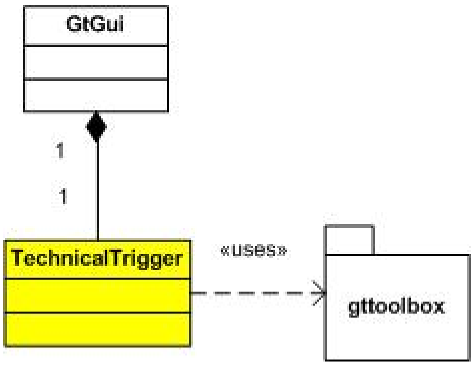

Figure 21: Relationships associated with class TechnicalTrigger.

4.2.5 PtcPanel

The class `PtcPanel` (see Figure 22) is contained in the `GtGui` class. This class is used to control and monitor the Partition Controller board of a given TTC/DAQ partition.
This class uses objects of the classes `PtcDialogSetPartition`, `FdlMaskDialog` and `PtcMonitorCounters`.

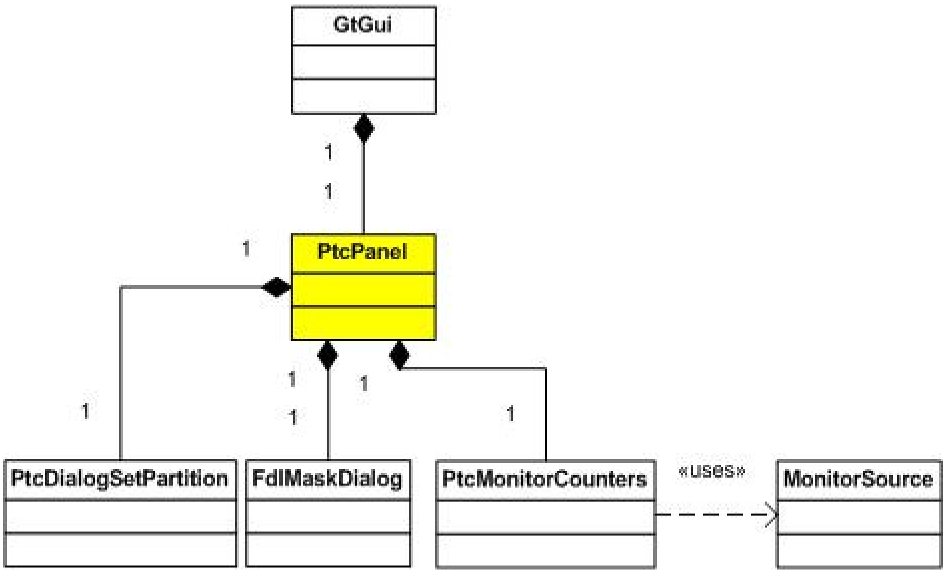

Figure 22: Relationships associated with class PtcPanel.

4.2.6 MasterDialogApply

The class `MasterDialogApply` (see Figure 23) is part of the class `MasterPanel`. This class is used to apply the actual configuration either to hardware or to the configuration DB. For now it is just a placeholder, since applying to the hardware happens immediately after an event like a button click has been triggered. Applying of the whole configuration to the database will be implemented in a further step.
The visual content comprises a button. Next to the button there exists a list with the available choices for applying the changes to hardware or database.

The behavior is encapsulated by the `OnClick` event. It is triggered by clicking on the “OK” or the “Cancel” button.

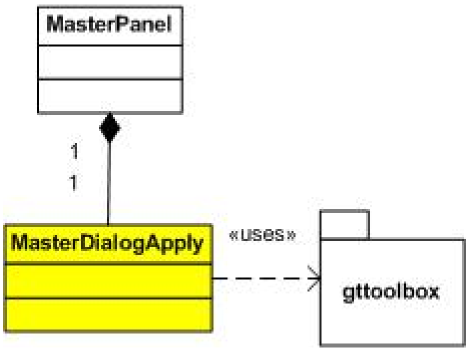

Figure 23: Relationships associated with class MasterDialogApply.

4.2.7 MasterDialogLoad

The class `MasterDialogLoad` (see Figure 24) is contained in the `MasterPanel` class. This class is used to load the actual configuration from the hardware or from the Configuration DB to the Control Panel. For now it is just a placeholder, since loading from the hardware is done by default. Loading of the whole configuration from the database will be implemented in a further step.

The visual content comprises a button. Next to the button there exists a list with the available choices for getting the information from hardware or database.

The behavior is encapsulated by the `OnClick` event. It is triggered by clicking on the "OK" or the "Cancel" button.

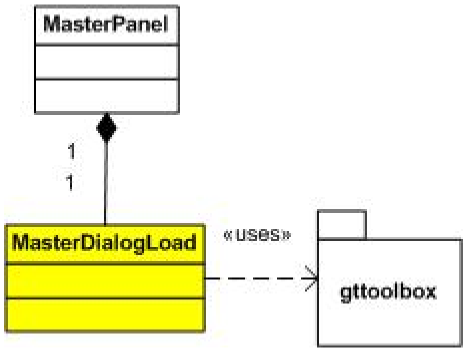

Figure 24: Relationships associated with class MasterDialogLoad.

4.2.8 MasterTimeSlots

The class `MasterTimeSlot` (see Figure 25) is contained by `MasterPanel`. This class is used to visualize and modify the time slots assigned to each TTC/DAQ partition.

The visual content comprises a table with the TTC/DAQ Partition numbers and eight `InputBoxes` below them. The `InputBoxes` show the value of their corresponding time slot. A common button is provided for applying the whole table to the hardware.

The behavior is encapsulated by the `OnSubmit` event of the form that contains the different `InputBoxes`.

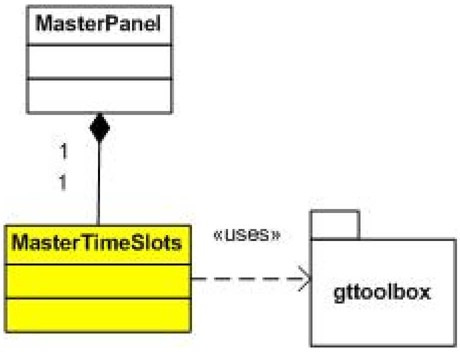

Figure 25: Relationships associated with class MasterTimeSlots.

4.2.9 PartitionGraph

The class `PartitionGraph` (see Figure 26) is contained in the `MasterPanel` class. This class is used to visualize the assignment of subdetectors to a given TTC/DAQ Partition. The assignment is stored in the struct `PtcDetectorStruct`.
The visual content comprises a matrix with 32x8 fields, which describes the assignment of the TCC Subdetectors to its TCC/DAQ Partitions and to its status. The assigned field is marked either green or red depending on if the enable-bit is set 1 or 0.

The behavior is encapsulated by one event, namely the `OnClick` event. Its purpose is to refresh the graph.

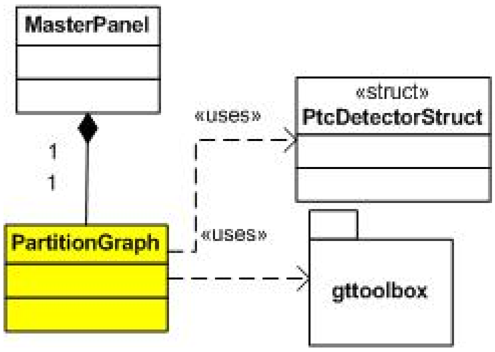

Figure 26: Relationships associated with class PartitionGraph.

4.2.10 PtcDialogSetPartition

The class `PtcDialogSetPartition` (see Figure 27) is contained in the `PtcPanel` class. This class is used to visualize and modify the enable bit of a TTC/DAQ partition for an assigned subdetector. Only already previously assigned subdetectors can be enabled or disabled.

The event used for enabling and disabling the subdetectors is the `OnSubmit` event. By executing this event, the submitted values are applied to the hardware.

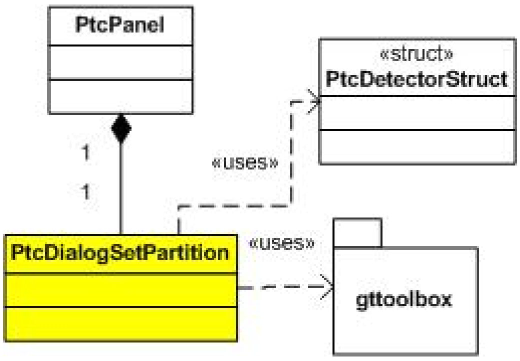

Figure 27: Relationships associated with class PtcDialogSetPartition.

4.2.11 PtcMonitorCounters

The class `PtcMonitorCounters` (see Figure 28) is contained in the `PtcPanel` class. This class is used to retrieve the Monitor Counters and to visualize them in an appropriate format. There exist 40 Monitor Counter bits, which contain the status of the hardware.

The event used in that class is `OnClick`. It allows the refreshing of the Monitor Counters.

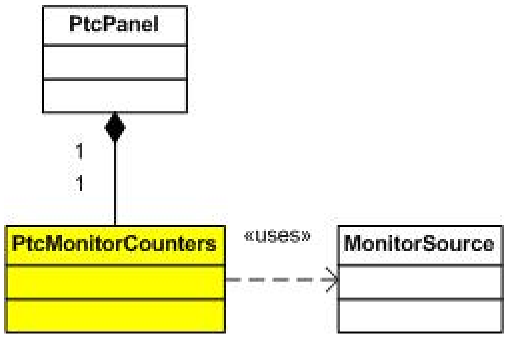

Figure 28: Relationships associated with class PtcMonitorCounters.

4.2.12 FdlMaskDialog

The class `FdlMaskDialog` (see Figure 29) is contained in the `PtcPanel` class. This class is used to define the Final Or Mask for a given TTC/DAQ partition.

The `OnSubmit` event is used to apply the value of the `CheckBoxes` to the hardware.

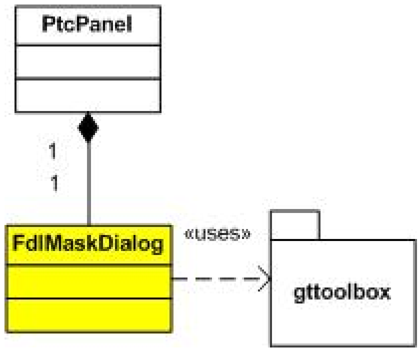

Figure 29: Relationships associated with class FdlMaskDialog.

4.2.13 PtcDetectorStruct

The structure `PtcDetectorStruct` (see Figure 30) is used by the `PtcDialogSetPartition` and `PartitionGraph` classes in order to store the enable bit and the assigned subdetector of a given TTC/DAQ partition.

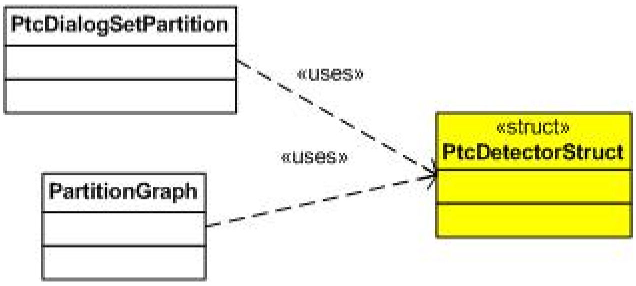

Figure 30: Relationships associated with struct PtcDetectorStruct.

4.2.14 MonitorSource

The class `MonitorSource` is a descendant of `DataSource` (a class of the TS framework) used to define the monitorable items of the GT Cell.

Actually the Monitor Counters of TCS are available through the class `MonitorSource` (see Figure 31).

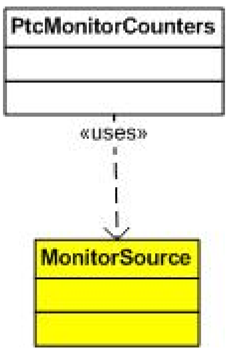

Figure 31: Relationships associated with class MonitorSource.

4.3 GT Cell

As has been shown in section 3.1 the GT Control Panel has been developed within a layered architecture. Therefore, the GT Control Panel is decoupled from the other software layers and can be developed separately. In order to use the GT `CellCommands` the namespace `gttoolbox` has been introduced. Every function in this namespace uses a `CellCommand` descendant that has been implemented by the GT developers in order to create a SOAP/C++ interface to the Online Software Infrastructure. All the functions within the `gttoolbox` namespace interact with the VME controller in the GT Crate.

Table 1 shows the relation between the different layers for the GT software. Every listed `CellPanel` descendant uses one or several `CellCommands` through the wrapper provided by the `gttoolbox` namespace.

Class	**FUNCTIONALITY within GT Control Panel**	**gttoolbox command**	**GT Cell command**
MasterDialogLoad	displaying the assignment in the graph	getAssignPart	GtTcsMasterGetAssignPart
gttoolbox, MasterDialogLoad	displaying the assignment in the graph	getAssignPartEn	GtTcsMasterGetAssignPartEn
PtcDialogSetPartition, MasterDialogApply	setting the enable bit for the subdetector to partition assignment	setAssignPartEn	GtTcsMasterSetAssignPartEn
PtcPanel	displaying the random trigger frequency	getRndFrequ	GtTcsPtcGetRndFrequ
PtcPanel	setting the random trigger frequency	setRndFrequ	GtTcsPtcRndFrequ
PtcPanel	starting the random trigger	startRndTrigger	GtTcsPtcStartRndTrigger
PtcPanel	stopping the random trigger	stopRndTrigger	GtTcsPtcStopRndTrigger
MasterTimeSlots	displaying time slots	getTimeSlot	GtTcsPtcGetTimeSlot
MasterTriggerMenu, TechnicalTrigger	displaying rate counters	readRateCounter	GtFdlReadRateCounter
MasterTriggerMenu, TechnicalTrigger	setting the prescales	setPrescaleFactor	GtFdlSetPrescaleFactor
MasterTriggerMenu,	displaying the prescales	getPrescaleFactor	GtFdlGetPrescaleFactor

TechnicalTrigger			
PtcPanel	reseting the partition	hardReset	GtTcsPtcHwReset
PtcPanel	resynchronizing the partition	resync	GtTcsPtcResync
PtcPanel	setting the FinOrMask	setFinOrMask	GtFdlSetFinOrMask
MasterTimeSlots	setting the time slots	setTimeSlot	GtTcsPtcSetTimeSlot
PtcPanel	starting the run	startRun	GtTcsPtcStartRun
PtcPanel	stopping the run	stopRun	GtTcsPtcStopRun
PtcPanel	displaying the FinOrMask	getFinOrMask	GtFdlGetFinOrMask
PtcPanel	displaying the partition input status	getMasterPtcInputStat	GtTcsMasterPtcInputStat
PtcPanel	displaying the partition output status	getMasterPtcOutputStat	GtTcsMasterPtcOutputStat
PtcDialogSetPartition, PartitionGraph	displaying detector status	getMasterDetectorStat	GtTcsMasterGetDetectorStat
MasterTriggerMenu	used for creating the trigger menu	getNumberOfAlgos	GtFdlGetNumberOfAlgos
TechnicelTrigger, MasterTriggerMenu	used for creating the technical trigger menu	getNumberOfTechTrigge rs	GtFdlGetNumberOfTechTrigg ers

Table 1: Relation between different layers of the GT software architecture.

5 Testing

Chapter 5 describes the hardware set-up and the hardware testing procedure. The testing procedure includes the tests with dummy (see information about HAL library [32]) and real hardware. The tests have to be seen in the context of the Magnetic Test and Cosmic Challenge (MTCC) effort. MTCC is the name for the test procedure during which the magnetic solenoid was activated and used with cosmic muons. The GT Control Panel tests have been designed with the help of a hardware expert.

5.1 Hardware Set-Up

The computer named `pcvahcms` (inside the `cmsdaqpreseries` network) has been used as the host for the GT Control Cell. It is connected to the GT hardware through a PCI board that communicates with the CAEN controller in the VME crate of the GT. That computer is located in the control room of the MTCCII (Green Barrack, Cessy, France). `pcvahcms` has been operated remotely using a web browser from computers inside the `cern` network in the building 40 at CERN (Meyrin, France).

`pcvahcms` is a i686 processor with 1 GB of memory and 50 GB hard disk. The operating system used is Scientific Linux (SL) CERN Release 3.0.6 [33]. The software installed is XDAQ 3.5.2, Power Pack 1.4.3 and Work Suite 1.5.1 [34]. The CAEN driver [35] used is V2718 PCI-VME Bridge v2.4. The Concurrent Versioning System (CVS) head of the TS Framework is being used during all the development (roughly corresponding to the version 1.3 of the framework).

5.2 Testing Procedure

For the testing procedure of the GT Control Panel tests with dummy and real hardware have been performed.

The dummy tests are useful in order to test the functionalities, the look and feel, and the integration of the GT Control Panel within the TS Framework. The hardware tests have been done in order to test the interaction of the GT Control Panel with the real GT hardware.

The goal of the hardware tests is to check if the functionalities of the GT Control Panel correspond to the desired hardware behavior. In order to test the proper functionality an expert is needed to support the testing procedure. The expert is responsible for the maintenance of the hardware set-up and he knows how to check (usually by using standalone tools) the hardware for correct behavior.

The procedure to compile, execute and test the control panel needs a number of steps which are explained in the sections 5.2.1- 5.2.6.

5.2.1 Login to cmsdaqpreseries Network

First the user has to login to the `cmsdaquser0` computer or other head-nodes of the `cmsdaqpreseries` network.

```
user@computer > ssh cmsdaquser0
```

Then he has to switch to the `/home/gtts` directory.

```
[cmsdaquser0] /home/user > cd /home/gtts
```

After that he has to create a new user group and change his rights by using the following commands:

```
[cmsdaquser0] /home/gtts > newgrp gtts
[cmsdaquser0] /home/gtts > umask 002
```

Finally the user has to retrieve the system environment variables by calling

```
[cmsdaquser0] /home/gtts > source environment
```

5.2.2 Compiling the Trigger

After being logged into the `cmsdaquser0` computer the user needs to compile the trigger branch of the software. In order to do that he has to change to the `/TriDAS/trigger` directory by executing the following commands.

```
[cmsdaquser0] /home/gtts > cd TriDAS/trigger/
[cmsdaquser0] /home/gtts/TriDAS/trigger > make Set=trigger
```

5.2.3 Login to the pcvahcms Computer

Before compiling the GT Cell the user has to connect to the `pcvahcms` test computer by calling

```
[cmsdaquser0] /home/gtts/TriDAS/trigger > ssh pcvahcms
```

After changing to the directory `/home/gtts` following commands have to be executed in order to create a new group and to change the user rights

```
[pcvahcms] /home/gtts > newgrp gtts

[pcvahcms] /home/gtts > umask 002
```

Finally the system environment variables from `pcvahcms` have to be loaded by executing

```
[pcvahcms] /home/gtts > source environment
```

5.2.4 Compiling the GT Cell

After having loaded the system environment variables the GT Cell can be compiled. Depending on if the GT Cell will be run in test or in hardware mode the following command has to be executed in the directory `TriDAS/trigger/gt` in order to change the system variable which stores information about the GTCAEN driver.

For dummy test:

```
[pcvahcms] /home/gtts/TriDAS/trigger/gt > export GTCAEN=no
```

For hardware test:

```
[pcvahcms] /home/gtts/TriDAS/trigger/gt > export GTCAEN=YES
```

Having set the GTCAEN variable the GT Cell has to be compiled by executing the following command:

```
[pcvahcms] /home/gtts/TriDAS/trigger/gt > make Set = gt
```

5.2.5 Launching the GT Control Panel

After having compiled the GT Cell the GT Control Panel application can be launched.

Therefore the monitor collector and the GT Control Panel have to be started in the directory /home/gtts/TriDAS/trigger/gt/ts/client

```
[pcvahcms]       /home/gtts/TriDAS/trigger/gt/ts/client >
         xdaq.sh -c xml/gtgui.xml -h cmsdaquser.pcvahpreseries -p 3666
[pcvahcms]       /home/gtts/TriDAS/trigger/gt/ts/client >
         xdaq.sh -c xml/gtgui.xml -h cmsdaquser.pcvahpreseries -p 3328
```

5.2.6 Accessing the Standalone GUI and other Tools connected to the Hardware

Having performed these steps the GT Control Panel can be used to operate and monitor the GT. To check if the commands sent by the GT Control Panel apply to the hardware properly, a set of tools is provided by a command based Standalone FDL GUI, a command based Standalone TCS GUI and a Java-implemented Standalone TCS GUI.

In order to access these tools a connection to the testing account of pcvahcms has to be established by executing

```
ssh testing@pcvahcms
pwd: ******
```

After having established the connection the timing module has to be started.

```
[pcvahcms] /home/testing > tim6uv2_gui
```

Afterwards CAEN has to be clicked within the launched GUI. The sections 4.2.6.1 and 4.2.6.2 describe the three possibilities on how to check the GT Control Panel functionality while interacting with the hardware.

5.2.6.1 Java-implemented Standalone TCS GUI

The GT, FDL, TCS etc. can be monitored and operated by launching the TCS Standalone GUI.

```
[pcvahcms] /home/testing > tcs9u_gui
```

The TCS Standalone GUI can be run in hardware mode or in dummy mode. The hardware mode allows the comparison between commands executed by means of the GT Control Panel and commands executed by the TCS Standalone GUI.

Note: In case the GUI crashes after clicking on CAEN the following command has to be executed

```
[pcvahcms] /home/testing > reloadit
```

This works only if the user is member of the sudoer list (`/etc/sudoers`).

5.2.6.2 FDL and TCS Standalone GUI's

GT, FDL and TCS can also be monitored by command line standalone applications. For starting the GT standalone applications the following command needs to be executed and slot 11 chosen:

```
[pcvahcms] /home/testing > gtl9u_standalone
```

For starting the FDL standalone applications following command needs to be executed and init board 1 chosen:

```
[pcvahcms] /home/testing > fdl9u_standalone and init board 1
```

6 Performance Measurements

Chapter 6 describes the procedure for measuring the loading time of individual widgets. It also depicts optimization steps performed during the development phase.

With an increased amount of widgets the GT Control Panel started reacting relatively slow. It has been possible to reach an acceptable level of responsiveness. Of course, it is still possible to improve it through a more careful design of the Control Panel and an extended measurement and optimization of the AjaXell library [36]; however a trade-off between the master project timeline and the Control Panel requirements has been reached. For example the Trigger Menu tab originally had 128 forms containing a `SubmitButton` and a `TextBox`. It took more than 13 seconds to load the changes in the web page. After measuring the performance associated to each widget a better solution has been found by trial and error and the response time has been decreased to 3 seconds. Upon reaching an acceptable level of responsiveness with a given functionality the improvement process was stopped. A means to measure the responsiveness of the Control Panel has been developed. From these measurements penalties associated to each widget have been estimated and this has allowed a factorization of the solution with a better response time. From these measurements it has also been possible to extract some optimization rules for improving the responsiveness of applications using the AjaXell library.

6.1 Procedure for Loading-Time Measurement

To measure the loading time of particular widgets some adaptations within the AjaXell library had to be made. All AjaXell widgets are available through the TriDAS CVS repository at `TriDAS/trigger/ts/ajaxell/src/common`. One has to edit the widgets one wants to measure and look for the third occurrence of `getResultType(Eventable::[onCorrespondingEvent])` within the AjaXell widget class.

Substitute the line

```
out << "response.setContent(data);" << std::endl;
```

by

```
out << "var start = new Date().getTime();" << std::endl;
out << "response.setContent(data);" << std::endl;
```

```
out << “var end = new Date().getTime();” << std::endl;
out << “alert(end - start);” << std:endl;
```

This has to be done for every widget one wants to measure. Afterwards, the AjaXell library has to be recompiled and the GT Cell has to be restarted. After executing the adapted event the user will receive an alert in the web browser indicating the amount of milliseconds it took to load the executed event.

6.2 Optimization Step 1: Reduce the number of Forms

Having measured the response time in the Trigger Menu (see Figure 32) for a different set of events, and assuming linear dependency a mean was needed to evaluate the average response time associated to each widget.

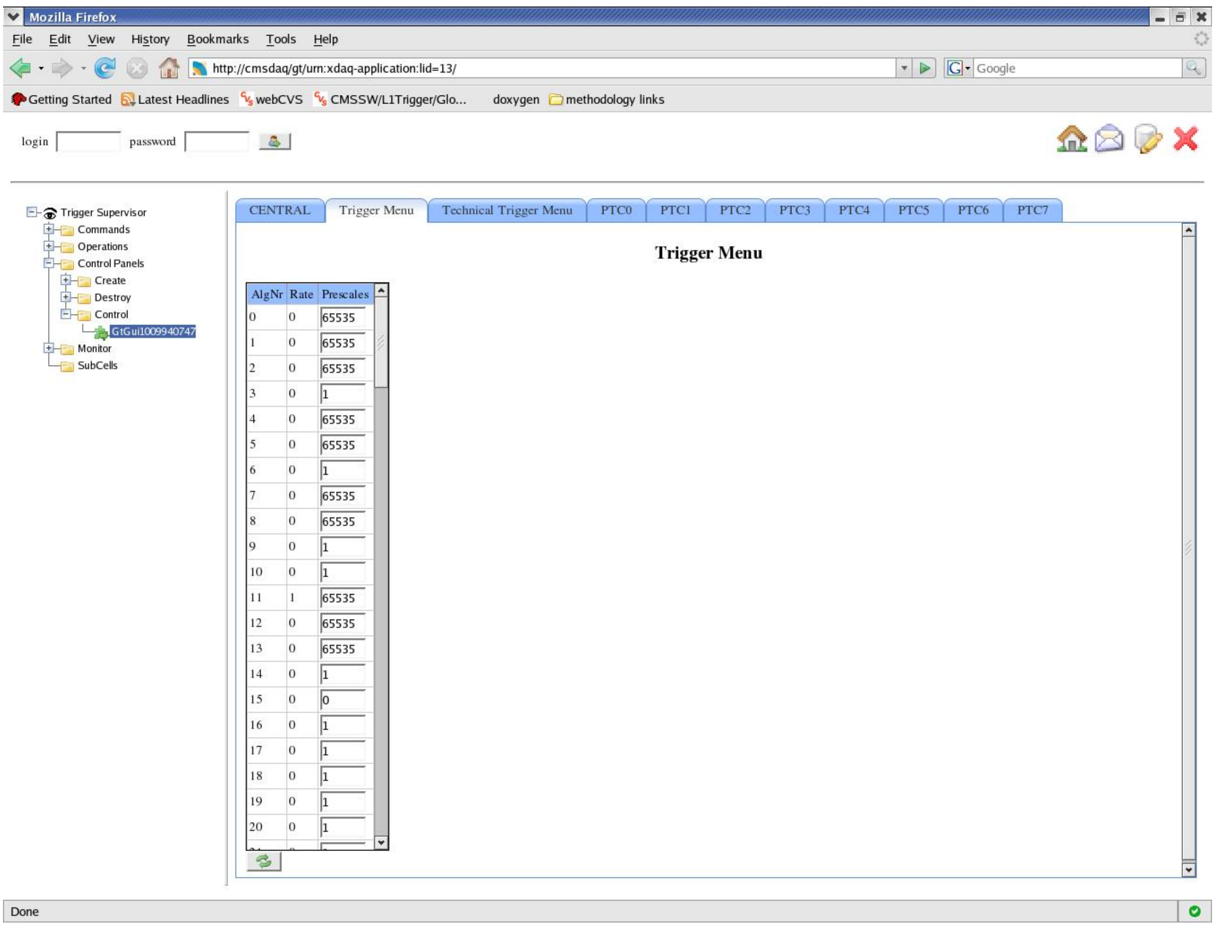

Figure 32: The Trigger Menu Tab which was separated from the CENTRAL Tab during one optimization step.

The most suitable and easiest to use function appeared to be the Linear Estimator (LINEST) function in MS Excel. Table 2 shows the measurements performed. The columns show the items that are sent back to the browser after an `XMLHttpRequest` [37] and the number of milliseconds that takes the modification of the Document Object Model (DOM) [38] tree and JavaScript objects to load. Each row represents a different look and feel for the same tab. In order to extract the response time penalty associated to each widget the linear estimator has been applied to the Table 2 (the MS Excel function `linest` was used).

Form [# Widgets]	**Submit event [# events]**	**simple widget [# Widgets]**	**Simple event [# event]**	**Total [ms]**
128	128	256	0	14000
128	0	256	0	3200
0	0	128	0	1000
0	0	128	128	1500
0	0	128	0	1100
128	0	0	0	1100

Table 2: Measurements performed for a different set of widgets.

The result is shown in Table 3. The first line gives the additional response time associated to each object sent back to the browser; the second line gives the standard error associated with the previous estimation [39].

m_iobind [ms]	**m_simple [ms]**	**m_formiobind [ms]**	**m_form [ms]**	**[ms]**
3,515625	8,203125	84,375	8,59375	1,42109E-13
0,676582347	0,390625	0,78125	0,552427173	70,71067812

Table 3: LINEST function to evaluate missing variables.

From the previous table it is possible to say that the major penalty in the response time comes from the `OnSubmit` event of a form. Therefore, if one wants to increase the performance of the Control Panel without modifying the AjaXell library the number of forms (and the corresponding `OnSubmit` event) which is sent back to the web browser has to be reduced. All these events can be substituted by events of simple widgets, displaying only a partial list of the algorithms, or using a single form.

6.3 Optimization Step 2: Reduce the number of Widgets and Events per XMLHttpRequest

As we have seen before the response time of the `OnFocus` event of the `Tab` widget depends on the number of widgets. This means that the response time is proportional to the number of widgets. Therefore, in order to improve the response time more tabs can be introduced in order to avoid complex refreshes of the DOM tree and of the JavaScript objects of the web browser.

In the first iteration, the Trigger Menu was part of the "CENTRAL" tab of the GT Control Panel. After recognizing the dependency of the response time on the number of widgets, the Trigger Menu was moved to another tab and the response time for both times has decreased by 10 s.

6.4 Optimization Step 3: Reduce the number of Widgets and use plain Text if possible

Using the same methodology as in section 5.2 it has been figured out that using plain Hypertext Markup Language (HTML) without JavaScript decreases the response time. In this subsection the measurements were done by trying to reduce the response time of the `PartitionGraph` (see Table 4 and Table 5).

Radiobutton [# Widgets]	**dialog [# Widgets]**	**div [# Widgets]**	**Total [ms]**
256	2	0	3700
256	1	0	3400
0	1	256	3800

Table 4: Performance measurements for a different set of Widgets that are sent back to the browser after clicking the refresh button of the Partition Graph.

Table 5 shows the result of the linear estimator using MS Excel. The first line gives the additional response time associated to each object which is sent back to the browser; the second line gives the standard error associated with the previous estimation.

Div [ms]	Dialog [ms]	Radiobutton [ms]	[ms]
0	300	-1,5625	3500

Table 5: Linest function to evaluate missing variables.

Originally, the `PartitionGraph` contained a matrix with active and inactive `RadioButtons`. Instead of `RadioButtons` splayed green or red – depending on the enable status – `PlainHtml` elements (which are `<DIV>` elements) are displayed. Therefore after executing the `XMLHttpRequest` less JavaScript/HTML/CSS [40] code is sent to the browser which leads to a performance improvement. Figure 33 shows the graph with green and red <DIV> elements.

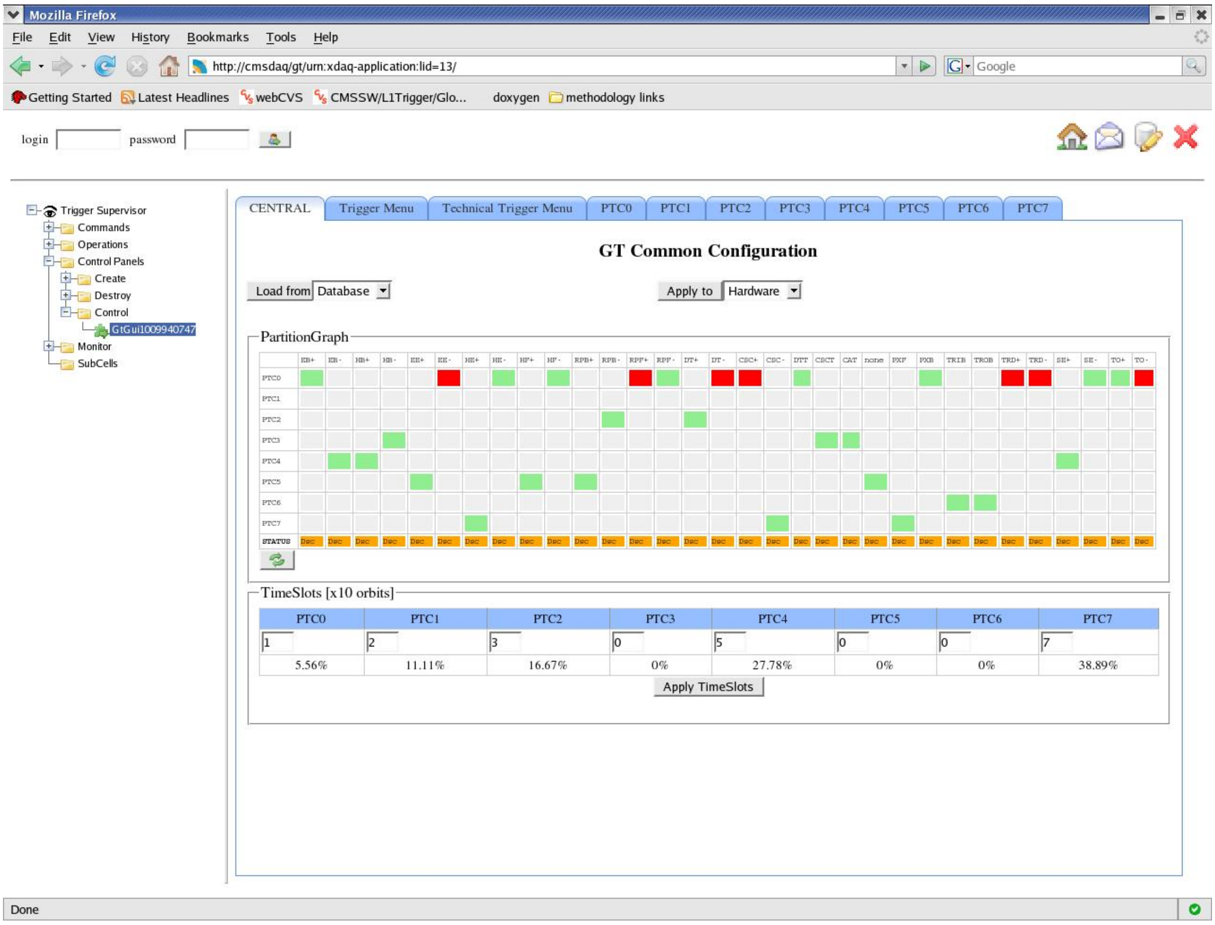

Figure 33: The PartitonGraph after the change filled with green and red PlainHtml <DIV> elements.

7 Trigger Supervisor Framework Integration

Chapter 7 describes the TS software infrastructure and the integration of the GT Control Panel into the Generic TS GUI. Thereby the Generic TS GUI, the TS Framework architecture, AjaXell and the most relevant class for creating Control Panels, namely CellPanel is being explained.

7.1 Generic TS GUI

The first project fully developed with the help of the AjaXell library was the Generic TS GUI, which had existed before as a website being developed without the use of AjaXell. The Generic TS GUI integrates services offered by the TS in one single web application in order to monitor, operate, setup and test the L1T of the CMS experiment, to populate the Configuration Database for any L1T subsystem and to access the support, documentation and the logging records for audit trials and post-mortem analysis. The visual difference between the old Generic TS GUI developed without AjaXell and the new one is depicted in Figure 34. While the old GUI was browsed by means of hyperlinks to the different sections, the new GUI provides trees and tabs which enable the enhanced visualization of a higher amount of information. Originally in the old GUI a whole webpage had to be refreshed in order to visualize the information retrieved from the server. The information provided in the new GUI can be refreshed merely in a specific part of the web page which improves the loading time.

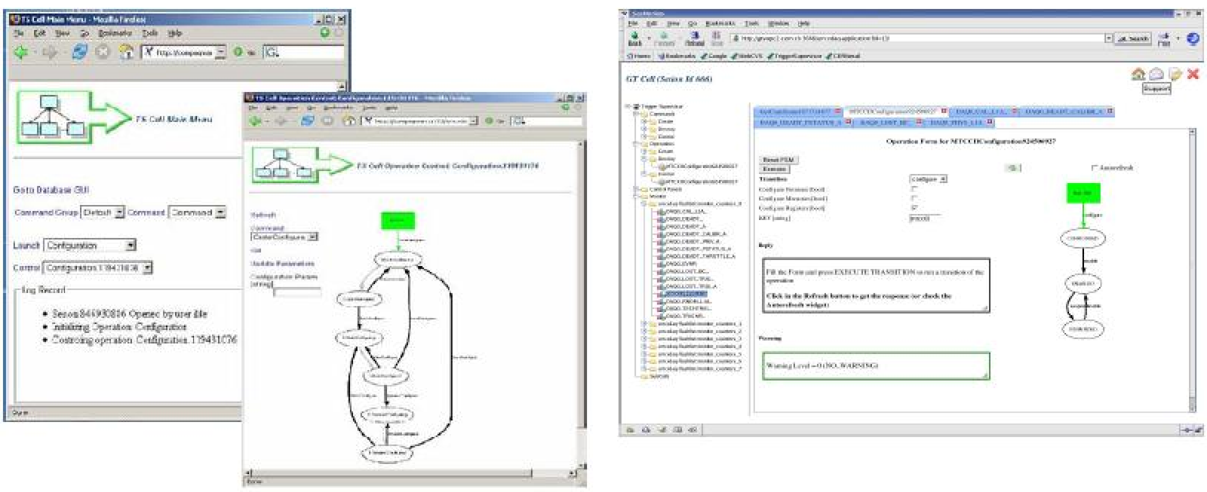

Figure 34: The two web pages on the left show the old Generic TS GUI which was loaded as a whole at one glance. The webpage on the right contains tabs, which are refreshed separately.

7.2 Architecture Overview

Each Control Panel is part of the TS Framework. Figure 35 shows the dependencies between the `CellPanel` class and the classes which instantiate and use the `CellPanel` class. It also clarifies that the author

The `CellPanel` class is the mother class for creating Control Panels. Any Control Panel inherits from `CellPanel` which itself provides necessary methods for displaying and handling any `CellPanel`. First it contains the `getContext()` method, which is needed to retrieve the Uniform Resource Name (URN), Uniform Resource Locator (URL) and the specific session id of the Control Panel. For error handling it inherits the `getLogger()` method which retrieves the log4cplus object containing all log information. Additionally it contains the `layout()` method for setting the layout of the widgets and the `html()` method for printing the content of the Control Panel.

`CellPanel` inherits from `AutoHandledWidget`. This allows the Control Panels, which inherit from `CellPanel`, to be used themselves as containers and widgets including events.

In order to make the Control Panels accessible within the class `CellPage`, which corresponds to the Generic TS GUI, each Control Panel has to be added to the object `CellPanelFactory`. The `CellPanelFactory` itself is initialized by the `CellAbstractContext`.

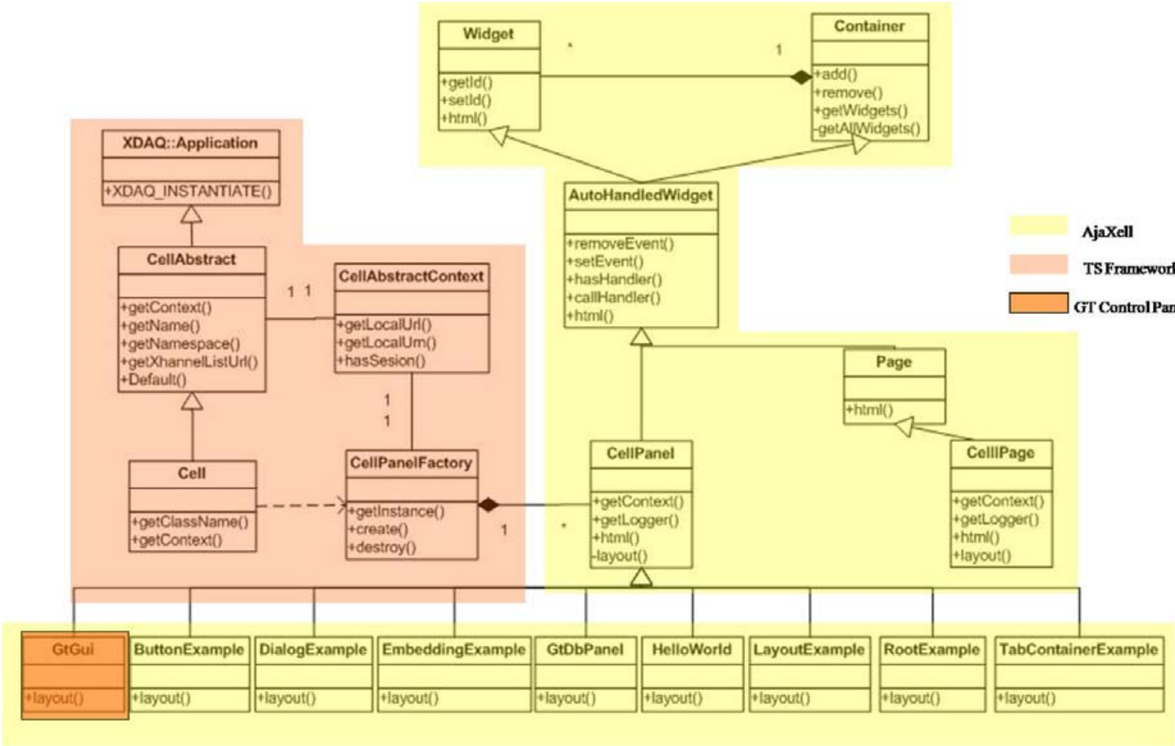

Figure 35: Class-Diagram showing the GT Control Panel and the dependencies within the TS Framework.

7.3 AjaXell Motivation

A series of L1T subsystem board level control panels needs to be integrated into the TS Framework. Originally these control panels have been developed as standalone tools (Tcl/Tk, C, C++, Qt, Java) being used locally or through remote terminals.

XDAQ version 3 provides a Hypertext Transfer Protocol (HTTP) engine compliant with the Common Gateway Interface (CGI), which allows the execution of applications within the web browser. The server side library used for the development of these Control Panels needed to support new widgets and event based behavior. During the evaluation process the most suitable library Wt [41] only supported FastCGI applications under Linux. Therefore a new Linux C++ CGI library named AjaXell (available as a SourceForge project [42]) has been developed fulfilling following requirements:

1. Implementation of an event based programming model.
2. Implementation of extended widgets.
3. Modular and extensible design.

4. C++ Linux compatibility.
5. Web browser independence.
6. CGI server independence.
7. Open Source solution.
8. Maintenance of commonly used technologies.

This library works within XDAQ [43] and within Apache Tomcat [44] applications opens doors for future projects.

For the client side there have been examined technologies like Java Applets, which use files with too large size to deal with XDAQ; XUL and XAML which are browser specific and therefore not fulfilling the application requirements; and Adobe Flash which comes with vendor lock-in and high learning costs. Finally AJAX was chosen.

The chosen technology AJAX provides following features:

1. HTML and Cascading Style Sheets (CSS).
2. Document Object Model (DOM).
3. XMLHttpRequest for asynchronous data exchange.
4. XML as format for data exchange.

The architecture of the AjaXell library is described in the following section.

7.4 AjaXell Architecture

This section explains the C++ library which is used in the GT Control Panel. The AjaXell library is based on AJAX, which allows sending asynchronous XMLHttpRequests to the server in order to refresh small parts of the website. Figure 36 depicts the process flow within the AjaXell library between the client being the browser and the server being a XDAQ application. The data exchange sequence follows six steps.

First a DOM event is triggered. This may be a click on a button, or a tick on a checkbox etc. Second the event is redirected to a JavaScript handler, which has been generated by AjaXell and loaded into the browser. Third the JavaScript handler generates and sends an XMLHttpRequest to the server passing a widget, the value and the event type. Fourth the passed parameters are read and redirected to the adequate C++ method which operates as an event handler. Fifth the C++ event handler creates HTML and JavaScript code corresponding to the request and sends it back to the browser. Finally the DOM JavaScript handler modifies the specific part of the website and refreshes the output.

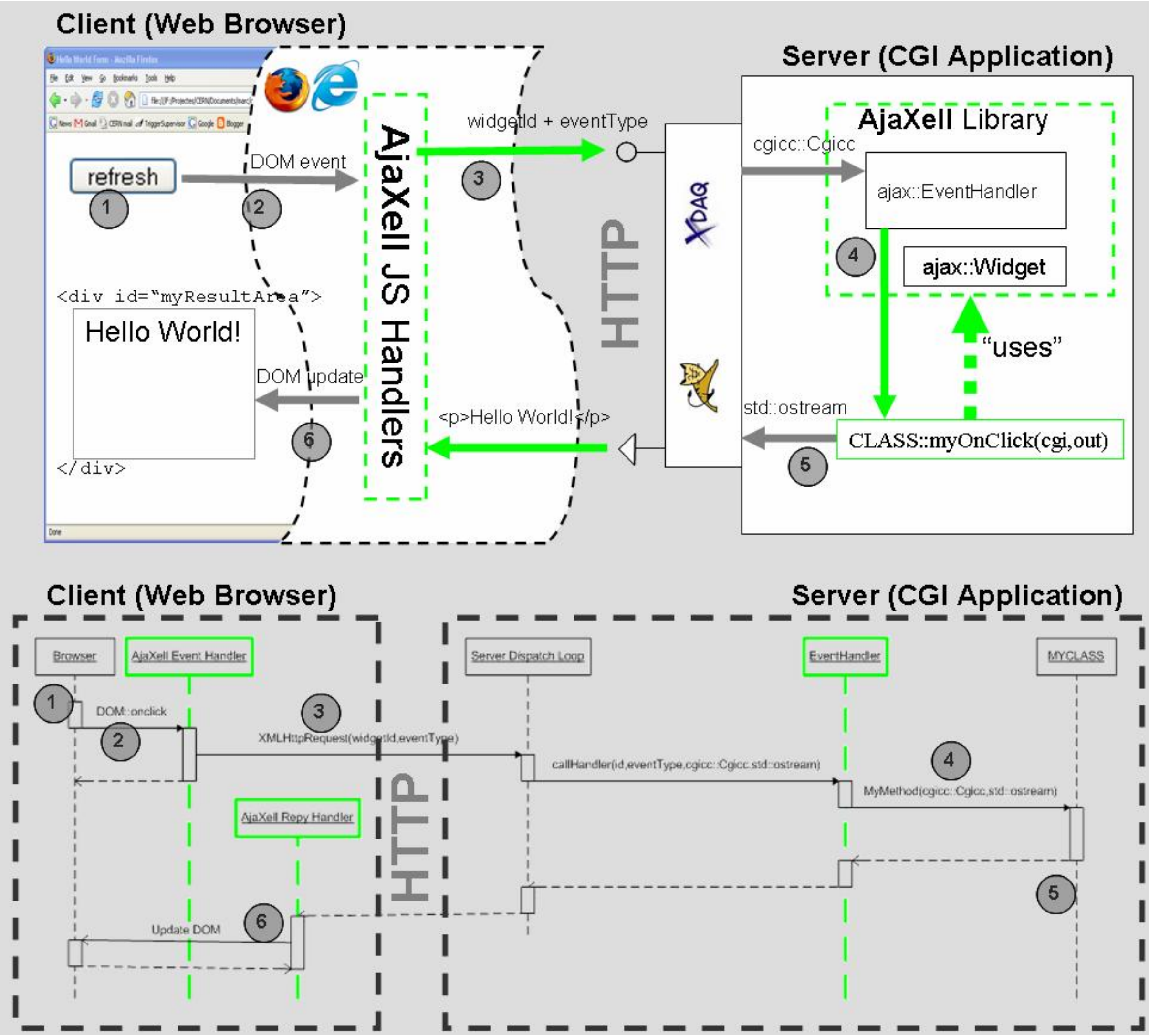

Figure 36: AjaXell Information Flow between Client and Server [36].

In order to understand the AjaXell library the classes `Widget`, `EventHandler` and `AutoHandledWidget` will be explained.

As shown in Figure 37 the `Widget` class is the root class of the library. It describes a widget, which can be any object within a browser. For example a button, a checkbox, a radio button etc. In order to display the widget within the browser the method `Widget::html` has to be called.

The class `AutoHandledWidget` gives the possibility to associate a widget with an event. Any descendant from `AutoHandledWidget` can therefore contain not only widgets itself, but also server side C++ event handlers, which are connected to the DOM event of the browser. The class `AutoHandledWidget` further on inherits from the class `Container`, which allows the `AutoHandledWidget` to contain several widgets as well as `AutoHandledWidgets` in a recursive way.

Finally the class `EventHandler` can be any C++ method having passed a Cgicc [45] object in order to parse the HTTP request. It dispatches the adequate callback for a given event type and identifier. The callback is written as result into a C++ standard output stream.

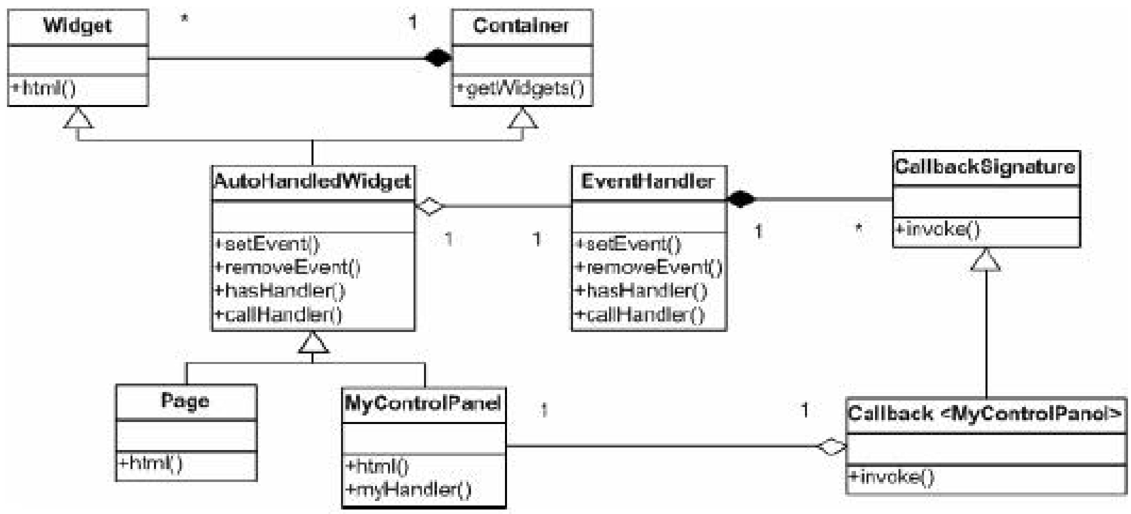

Figure 37: Relationship between Widget, Container, EventHandler and AutoHandledWidget.

7.5 CellPanel class

This section explains how a `CellPanel` is initialized and how it uses its methods. As mentioned in section 7.2 `CellPanel` is initialized by `CellPanelFactory` and inherits from `AutoHandledWidget` (see Figure 38).

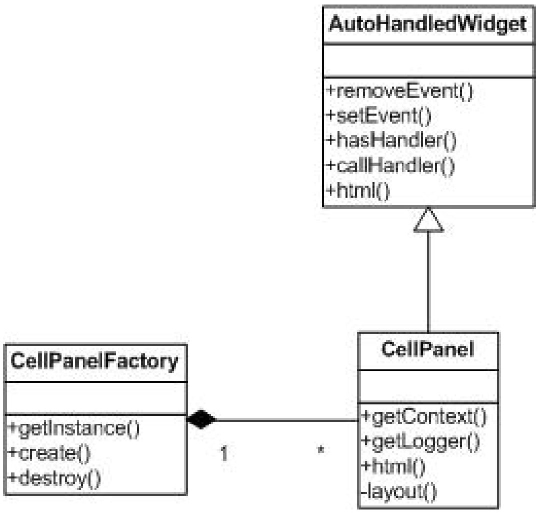

Figure 38: CellPanel dependency between CellPanelFactory, CellPanel and AutoHandledWidget.

The class `CellPanelFactory` stores, creates and destroys `CellPanels`. As the owner of the `CellPanels` the `CellPanelFactory` has three important methods for adding, removing and destroying a `CellPanel` instance within the `Cell` class.

```
void add(const std::string& name)

void remove();

void destroy(const std::string& instanceName);
```

There exist some more methods for getting the list of `CellPanel` classes, for getting a `CellPanel` instance and for creating a signature.

In order to add a `CellPanel` to the `Cell` one needs to add the line

```
getContext()->getPanelFactory()->add<panels::CellPanelName>("CellPanelName");
```

to the `Cell::init()` method. The Cell is a XDAQ application which corresponds to a node of a distributed system called TS System, which has a web interface.

The class `AutoHandledWidget` provides the methods (see Figure 39) for setting and removing an event and for retrieving and checking the corresponding handler. The three different `setEvent()` methods bind a handler, which passes a Cgicc-object and a standard output stream to an event. They are developed to serve three targets. The first `setEvent()` method has no target, the second `setEvent()` method sends the result to the `ResultBox` and the third `setEvent()` method sends the result to the `Dialog`. The methods `hasHandler()` and `removeHandler()` are used to check if a handler exists for a given widget.

```
void setEvent(ajax::Eventable* from, ajax::Eventable::EventType type, OBJECT* obj, void
   (OBJECT::*callback)(cgicc::Cgicc&,std::ostream&))void remove();

void setEvent(ajax::Eventable* from, ajax::Eventable::EventType type, ajax::ResultBox*
   to, OBJECT* obj, void (OBJECT::*callback)(cgicc::Cgicc&,std::ostream&))

void setEvent(ajax::Eventable* from, ajax::Eventable::EventType type, ajax::Dialog*
   dialog, OBJECT* obj, void (OBJECT::*callback)(cgicc::Cgicc&,std::ostream&))

bool hasHandler(const std::string& id, ajax::Eventable::EventType type);

void callHandler(cgicc::Cgicc& cgi, std::ostream& out);

void removeEvent(const std::string& id)

void html(cgicc::Cgicc& cgi, std::ostream& out);
```

Figure 39: Methods provided by class AutoHandledWidget.

The class `CellPanel`, which is the parent of any Control Panel, implements methods for displaying the Control Panels within one session providing a comprehensive debug possibility.

The method `getContext()` provides the URN, the URL, the `CellPanelFactory` for creating `CellPanels`, the `CellOperationFactory` for creating `CellOperations`, the `CellFactory` for creating commands and it conveys the session id for guarantying multiuser access to the `Cell`.

```
CellAbstractContext* getContext();
```

The method `getLogger()` provides a `log4cplus` object, which allows logging of errors and stack traces on different severity levels.

```
log4cplus::Logger& getLogger();
```

The method `html()` prints the html code for all widgets which are available in the current container.

```
void html(cgicc::Cgicc& cgi,std::ostream& out);
```

The method `layout()` is used to arrange widgets within the page.

```
virtual void layout(cgicc::Cgicc& cgi) = 0;
```

An example of how to bind an event to a widget is provided in Figure 40. The method `layout()` is used for adding and removing widgets and its events. Therefore first the layout has to be cleared by calling `remove()`. Then a `PlainHtml` widget is created and added to the panel. Afterwards a button is created with a caption and added to the panel. Finally an event is assigned by specifying the widget, the event, the result box, the current object and the event handler.

```
void panels::ButtonExample::layout(cgicc::Cgicc& cgi)

{

remove();

//Form title

ajax::PlainHtml* title = new ajax::PlainHtml();

title->getStream() << "<h4 align=\"center\">Control Panel Example: Button and the
   OnClick event</h4>" << std::endl;

add(title);

ajax::Button* button = new ajax::Button();

button->setCaption("click me!");

this->setEvent(button,ajax::Eventable::OnClick,result_,this,&ButtonExample::onClick);


add(button);

add(result_);

}
```

Figure 40: Binding an event to a widget with setEvent().

The `EventHandler` which reacts to the above specified event is depicted in Figure 41. The `EventHandler's` method name is `onClick()`. Within this method the result to be shown is created. In this case it is a `PlainHtml` widget containing html-code. To add the `PlainHtml` widget to the specified `ResultBox`, the `ResultBox` needs to be cleared by calling `remove()`. Afterwards the `PlainHtml` widget can be added. In order to display this widget on the client side one needs to call `html()`.

```
void panels::ButtonExample::onClick(cgicc::Cgicc& cgi,std::ostream& out)
{
   std::ostringstream msg;
   msg << "The ButtonExample::onClick method has been executed";

   ajax::PlainHtml* plain = new ajax::PlainHtml();
   plain->getStream() << "<p>" << msg.str() << "</p>";

   result_->remove();
   result_->add(plain);
   plain->html(cgi,out);
}
```

Figure 41: OnClick event example for clicking a button.

To adjust the layout of a widget the `LayoutElements` can be used, which have to be packed themselves into `LayoutContainer`. The `LayoutContainer` can be arranged differently. The possibilities offered are `LeftRight`, `TopBottom`, `None`. `LeftRight` means the `LayoutElements` will be arranged in the order they were added but assigning the left and right side with higher priority. `TopBottom` means the `LayoutElements` will be arranged by the same order as they were added but assigning the top and the bottom side with higher priority. None means, the elements will be assigned in spiral order.

```
void setLayout(ajax::LayoutContainer::Layout layout);
```

The `LayoutElements` themselves, which are put into the `LayoutContainer`, have an alignment method which is called `setPosition`. The possibilities offered are `Left`, `Right`, `Top`, `Bottom`, `Client`.

```
void setPosition(ajax::LayoutElement::Position position);
```

7.6 Other Subsystem Control Panels

In order to control and operate the L1T every subsystem can implement its own custom Control Panel with the help of the AjaXell library. Until now one custom control panel for the GT named "GT Control Panel" has been developed. Figure 42 shows how the GT Control Panel fits into the new Generic TS GUI. Within the Generic TS GUI an instance of the GT Control Panel has to be created by clicking on the "Create" leaf in the tree. Afterwards the GT Control Panel can be operated on the right side of the Page. Within the tree the GT Control Panel can be controlled and destroyed.

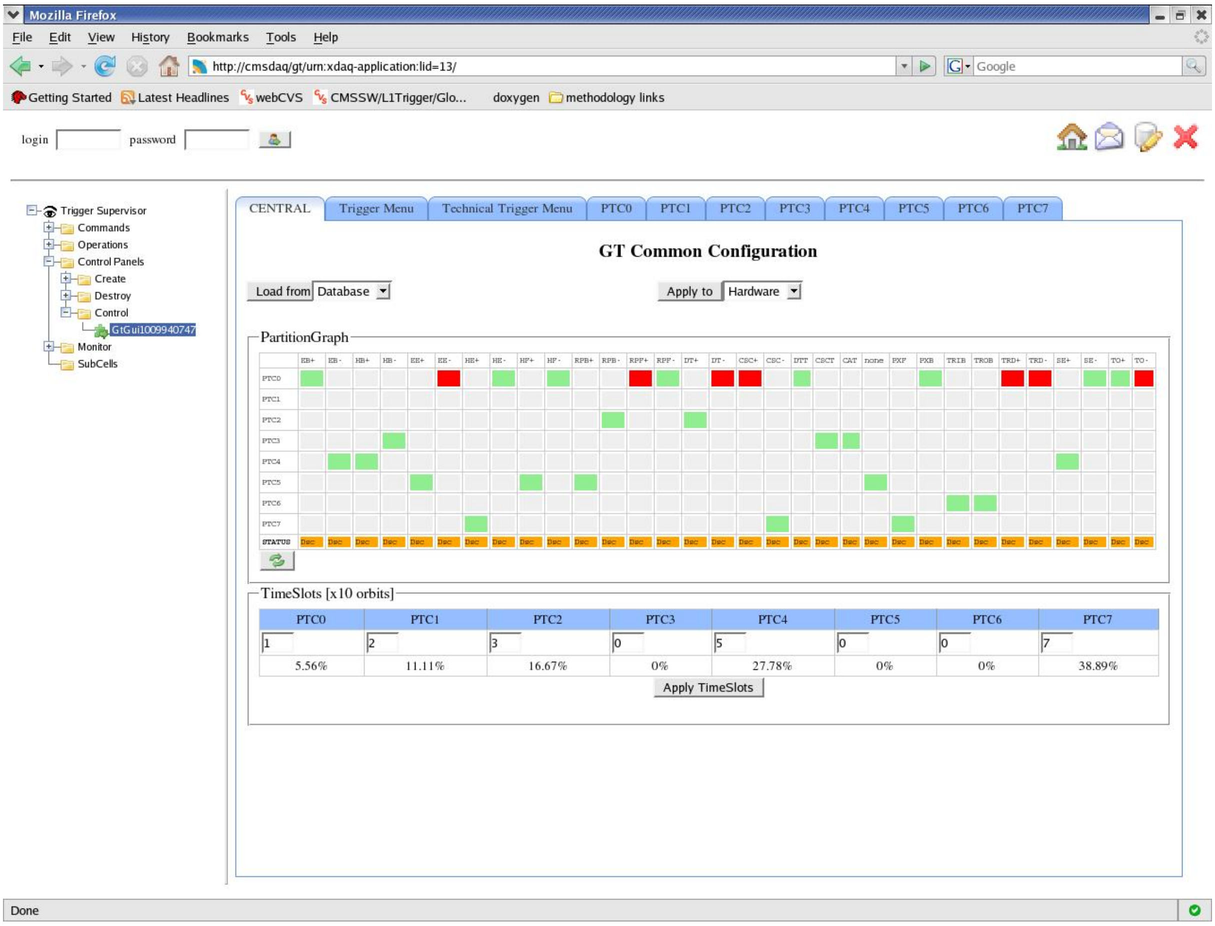

Figure 42: GT Control Panel initialized within the Generic GUI.

8 Conclusion

The Global Trigger Control Panel has been developed and integrated successfully whereby the concept of the Trigger Supervisor Control Panel Infrastructure was proven. Periodic usage during the commissioning run exercises and the ongoing development of new control panels for RCT, DTTF and GMT verify its success.

The functionalities implemented as well as the missing ones are shown in Table 6. Originally the Trigger Menu was intended to be developed completely. Therefore an XML parser for the conditions and names for the algorithms to be retrieved from the configuration database was needed. The development of a new parser or the decoupling of an existing XML parser being already used within the experiment would have exceeded the master project timeline.

The AjaXell library used for the master thesis project was found adequate with respect to the requirements of the GT Control Panel. A few discovered bugs were reported, which could be solved with the help of the AjaXell developers.

As can be seen in the table a second phase of development will be needed that will require the implementation of the full Trigger Menu and the configuration database tool, the integration with access control and the control of the trigger throttling rules.

Functionality		Status
Monitoring of	- TCS and FDL counters	Done
	- DAQ partition status	Done
	- Subdetector partition status	Done
Monitoring/Updating of	- Prescales	Done
	- Time Slots	Done
	- FDL Final OR Mask	Done
	- Partition Assignment	Done
	- Random Trigger Frequency	Done
Start Stop of	- Trigger	Done
	- Random Trigger	Done
	- Partition Assignment	Done
Reset and Resynchronization of	- PTC	Done
Control of	- Trigger Throttle Logic	To be Done
Creation of	- common and partition dependent keys	To be Done
Generation of	- Trigger Menu	To be Done
Generation of	- Configuration data base tool	To be Done
Implementation of	- Trigger Supervisor Access Control Integration	To be Done

Table 6: Implemented and missing Functionalities.

Acknowledgments

First I want to thank Marc Magrans de Abril. Thanks to his great support, patience, motivation and enthusiasm I was able to acquire all necessary knowledge for developing and integrating the GT Control Panel. I learned a lot from him about programming as well as about managing tasks and time. I also want to acknowledge the HEPHY group where I am deeply grateful to Claudia-Elisabeth Wulz and Ildefons Magrans de Abril for offering this project to me and for making it real. Due to Ildefons' know-how and project management skills I felt certain about the project's success whereby I could focus on the goals. For the testing phase I want to say thanks to Manfred Jeitler, who guided me through the hardware tests. Additionally I want to acknowledge Philipp Glaser, Ivan Mikulec, Barbara Neuherz, and Vasile Ghete for their help and advice.

Also I want to say thanks to my family which supported me during my studies and encouraged me to go to CERN.

Finally I want to say many thanks to Prof. Futschek for supporting my thesis at the Faculty for Software technology and interactive Systems E188 at Vienna University of Technology.

List of Figures

Abbreviations

API	Application Program Interface
CGI	Common Gateway Interface
CERN	Conseil Europ´een pour la Recherche Nucl´eaire
CMS	Compact Muon Solenoid
CSC	Cathode Strip Chamber
CSCTF	Cathode Strip Chamber Track Finder
CVS	Concurrent Versions System
DAQ	Data Acquisition
DB	Data Base
DOM	Document Object Model
DT	Drift Tube
DTTF	Drift Tube Track Finder
ECAL	Electromagnetic Calorimeter
FDL	Final Decision Logic
GMT	Global Muon Trigger
GTL	Global Trigger Logic
GUI	Graphic User Interface
HCAL	Hadronic Calorimeter
HF	Forward Hadronic Calorimeter
HLT	High Level Trigger
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSP	Java Server Pages
LHC	Large Hadron Collider
L1A	Level-1 Accept signal

L1T	Level-1 Trigger
OSWI	Online SoftWare Infrastructure
PSB	Pipeline Synchronize Buffer
RCMS	Run Control and Monitoring System
RPC	Resistive Plate Chamber
SMR	Session Manager
SOAP	Simple Object Access Protocol
TIM	Timing Board
TCS	Trigger Control System
TS	Trigger Supervisor
TTC	Timing, Trigger and Control System
TTS	Trigger Throttling Logic
URL	Uniform Resource Locator
URN	Uniform Resource Name
VME	VERSAmodule Eurocard Bus
XDAQ	Cross-Platform DAQ Framework
XML	Extensible Markup Language

Bibliography

[1] LHC Project Homepage, LHC - The Large Hadron Collider, http://lhc.web.cern.ch/lhc/ [15.4.2007]

[2] G. Altarelli, Proc. Large Hadron ColliderWorkshop, Aachen, 1990, eds. G. Jarlskog and D. Rein, CERN 90–10, vol. I, 153

[3] ATLAS Collaboration, ATLAS Technical Proposal, CERN/LHCC 1994-43 (1994)

[4] LHCb Collaboration, LHCb Technical Proposal, CERN/LHCC 1998-4 (1998)

[5] ALICE Collaboration, ALICE Technical Proposal, CERN/LHCC 1995-71 (1995)

[6] CMS Collarboration, The Compact Muon Solenoid - Technical proposal, CERN/LHCC 1994-38 (1994)

[7] CMS Physics Technical Design Report Volume I: Detector Performance and Software, CERN/LHCC 2006-001 (2006).

[8] L. Borello, E. Focardi, A. Macchiolo, A. Messineo, Sensor design for the CMS Silicon Strip Tracker, CMS. Note 2003/020 (2003).

[9] CMS Collaboration, The Electromagnetic Calorimeter Technical Design Report, CERN/LHCC 97-33 (1997)

[10] CMS Collaboration, The Hadron Calorimeter, Technical Design Report, CERN/LHCC 97-31 (1997)

[11] Letter of Intent for a General Purpose Detector at the LHC, Members of the CMS Collaboration, Della Negra et al., October 1992

[12] F.R. Cavallo, et al., Test of MB3 Muon Barrel Drift. Chambers with Cosmic Rays, CMS Note 2003/017

[13] Electromagnetic Secondaries and Punch-Through Effects in the CMS ME1/11. I. I. I. I. Belotelov, et al., CMS NOTE 2006/034

[14] CMS Collaboration, The TriDAS Project – Data Acquisition and High-Level Trigger Technical Design Report, CERN/LHCC 2002-26 (2002)

[15] CMS The TriDAS Project Technical Design Report, Volume 1: The Trigger Systems, CERN/LHCC 2000 – 38 CMS TDR 6.1 December 15, 2000.

[16] C.-E. Wulz: “Concept of the First Level Global Trigger for the CMS Experiment at LHC”, Nucl. Instr. Meth. A473/3 (2001) 227-238

[17] Philipp Glaser, Diploma Thesis March 20, 2007: Software Integration of the Global Trigger, University of Technology Vienna

[18] B. G. Taylor, TTC Distribution for LHC Detectors, IEEE Trans. Nucl. Sci. 45, No. 3 (1998) 821.]

[19] H. Bergauer, M. Padrta, A. Taurok; Trigger Control Module 9U-Version - TCS chip V0007, 2004, Hochenergie-Institut Wien

[20] H. Bergauer, K. Kastner, M. Padrta, A. Taurok, Global Trigger Logic Module 9U-Version, April 2006, Hochenergie-Institut Wien

[21] A. Racz et. al., "Trigger Throttling System for CMS DAQ", Sixth Workshop on Electronics for LHC Experiments, CERN/LHCC/2000-041

[22] H. Bergauer, K. Kastner, M. Padrta, A. Taurok, Timing Module in the Level 1 Global Trigger 6U-Version, August 2005,

[23] V. Brigljevic et al., Run Control and Monitor System for the CMS Experiment, Computing in High Energy and Nuclear Physics, 2003, La Jolla, California

[24] CMS NOTE 2005/011, I. Magrans, J. Varela, C.-E. Wulz, Concept of the CMS Trigger Supervisor, TNS-IEEE

[25] Trigger Supervisor Website, http://triggersupervisor.cern.ch/ [15.4.2007]

[26] J. Gutleber, S. Murray, L. Orsini, Towards a homogenous architecture for high-energy physics data acquisition systems, Comput. Phys. Commun. 153, Issue 2 (2003) 155 – 163

[27] SOAP Interface Specification, http://www.w3.org/TR/soap/ [14.4.2007]

[28] CGI Specification, http://www.w3.org/CGI/; [14.4.2007]

[29] M.Jeitler et al., The Level-1 Global Trigger for the CMS Experiment at LHC, Journal of Instrumentation (2007) P01006, http://stacks.iop.org/ 1748-0221/2/P01006

[30] Java Sun including Java Server Pages http://java.sun.com/ [14.4.2007]

[31] TS NOTE 2006/003 , P. Glaser, Noebauer T., Magrans I, Magrans M., Urank M., User's Guide Trigger Supervisor

[32] Christoph Schwick, Hardware Access Library http://cmsdoc.cern.ch/~cschwick/software/documentation/HAL/index.html [15.4.2007]

[33] Scientific Linux 3 CERN Release http://linux.web.cern.ch/linux/scientific3/ [15.4.2007]

[34] J. Gutleber, L. Orsini et al, "Using XDAQ in Application Scenarios of the CMS Experiment", Conference for Computing in High-Energy and Nuclear Physics, CERN 2003

[35] CAEN V2718 PCI-VME Optical Link Bridge, Version 2.4 http://www.caen.it/nuclear/product.php?mod=V2718 [15.4.2007]

[36] Ildefons Magrans de Abril, Marc Magrans de Abril, Enhancing the User Interface of the CMS Level 1 Trigger Online Software with Ajax, Real Time Conference, CERN 2007

[37] Anne van Kesteren, October 2007, The XMLHttpRequest Object Working Draft, http://www.w3.org/TR/XMLHttpRequest [18.1.2008]

[38] Document Object Model Level 2 HTML Specification, World Wide Web Consortium, January 2003 http://www.w3.org/DOM/ [18.1.2008]

[39] Microsoft Corporation, Online Help regarding the Linest function of MS Excel, http://office.microsoft.com/en-us/excel/HP100698381033.aspx [14.4.2007]

[40] Cascading Style Sheets W3C Specification, http://www.w3.org/Style/CSS/ [14.4.2007]

[41] Wt C++ Web Toolkit Project v. 1.99, http://sourceforge.net/projects/witty/ [14.4.2007]

[42] AjaXell Library Source Forge project, http://sourceforge.net/projects/AjaXell/ [14.4.2007]

[43] J. Gutleber, L. Orsini et al., “Hyperdaq – Where Data Adquisition Meets the Web”, 10th ICALEPCS, 10 - 14 Oct 2005

[44] Apache Tomcat Webserver; http://tomcat.apache.org/ [14.4.2007]

[45] CGICC Interface; http://www.gnu.org/software/cgicc/ [14.4.2007]

Appendix 1: Source Code

The source code I wrote for the master thesis project is available online.

- Header Files http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/TriDAS/trigger/gt/ts/cell/include/gtgui/?cvsroot=tridas
- Classes http://isscvs.cern.ch/cgi-bin/viewcvs-all.cgi/TriDAS/trigger/gt/ts/cell/src/common/?cvsroot=tridas

Headers	**Classes**
FdlMaskDialog.h	FdlMaskDialog.cc
GtGui.h	GtGui.cc
MasterDialogApply.h	MasterDialogApply.cc
MasterDialogLoad.h	MasterDialogLoad.cc
MasterDialogSetMask.h	MasterDialogSetMask.cc
MasterPanel.h	MasterPanel.cc
MasterTimeSlots.h	MasterTimeSlots.cc
MasterTriggerMenu.h	MasterTriggerMenu.cc
PartitionGraph.h	PartitionGraph.cc
PtcDialogSetPartition.h	PtcDialogSetPartition.cc
PtcMonitorCounters.h	PtcMonitorCounters.cc
PtcPanel.h	PtcPanel.cc
TechnicalTrigger.h	TechnicalTrigger.cc
gttoolbox.h	gttoolbox.cc
PtcDetectorStruct.h	

Table 7: Headers and classes developed within the scope of the master thesis project.