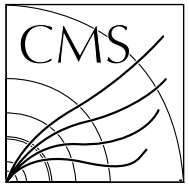
Available on CMS information server

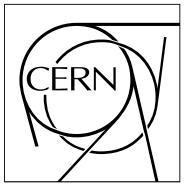
CMS NOTE 2005/011



The Compact Muon Solenoid Experiment

CMS Note

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



13 June 2005

Concept of the CMS Trigger Supervisor

Ildefons Magrans de Abril, Claudia-Elisabeth Wulz

Institute for High Energy Physics of the Austrian Academy of Sciences, Nikolsdorfergasse 18, A-1050 Vienna, Austria

João Varela

LIP - Lab. de Instrumentação e Física Exp. de Partículas, Lisboa, Portugal,
also with CERN, Geneva, Switzerland, and IST - Inst. Superior Técnico, Lisboa, Portugal

Abstract

The Trigger Supervisor is an online software system designed for the CMS experiment at CERN. Its purpose is to provide a framework to set up, test, operate and monitor the trigger components on one hand and to manage their interplay and the information exchange with the run control part of the data acquisition system on the other. The Trigger Supervisor is conceived to provide a simple and homogeneous client interface to the online software infrastructure of the trigger subsystems. This document specifies the functional and non-functional requirements, design and operational details, and the components that will be delivered in order to facilitate a smooth integration of the trigger software in the context of CMS.

Submitted to *IEEE Transactions of Nuclear Science*

Contents

1 Introduction

The experiment CMS (Compact Muon Solenoid) at the Large Hadron Collider (LHC) of CERN, the European Organization for Nuclear Research in Geneva, is a detector designed to find answers to some of the fundamental open questions in physics today [1].

The LHC provides counter-rotating proton beams with high energies, which are made to collide in the centre of CMS, producing a large number of particles. The trigger and the data acquisition are vital components of the experiment. The two systems are responsible for the selection and recording of collision events. Since millions of collisions occur each second and only a small fraction of these can provide insight into new physics, events have to be selected online according to their properties. The trigger system is composed of subsystems and is organized in two basic levels. The Level-1 Trigger (L1T) [2] consists of custom developed and largely programmable electronics, the High Level Trigger (HLT) is a large computer farm [3]. Apart from recording events, the data acquisition system (DAQ) [3] also provides the Run Control and Monitoring System (RCMS) framework [4] for the experiment.

1.1 The Trigger Supervisor

The purpose of the Trigger Supervisor (TS) is to set up, test, operate and monitor the trigger components on one hand and to manage their interplay and the information exchange with the run control and monitoring part of the data acquisition system on the other. It is conceived to provide a simple and homogeneous client interface to the online software infrastructure of the trigger subsystems. Facing a large number of trigger subsystems and potentially a highly heterogeneous environment resulting from different subsystem Application Program Interfaces (API), it is crucial to simplify the task of implementing and maintaining a client that allows operating several trigger subsystems either simultaneously or in standalone mode.

An intermediate node, lying between the client and the trigger subsystems, which offers a simplified API to perform control, monitoring and testing operations will simplify the design of this client. This layer provides a uniform interface to perform hardware configurations, monitor the hardware behavior or to run tests where several trigger subsystems participate. In addition, this layer coordinates the access of different users to the common trigger resources.

The operation of the trigger will necessarily be inside the broader context of the experiment operation. In this context, RCMS will be in charge to offer a control window from where an operator can run the experiment, and in particular the trigger system. On the other hand, it is also necessary to be able to operate the trigger system independently of the other experiment subsystems. This independence of the TS will be mainly required during the commissioning and maintenance phases.

The design of this hardware management system for a large experiment like CMS should cope with the following complexity dimensions:

- Number: O(10^3) hardware modules.
- Evolution: The preparation and operation of high-energy physics experiments typically spans over a period of many years (i.e. 1992, letter of intent for the CMS experiment [5]). During this time the hardware and software environments evolve (e.g., CDF Upgrade [6], D0 Upgrade [7]).
- Human: despite the necessary and highly hierarchic structure in a collaboration of more than 2000 people, different subgroups might implement solutions based on heterogeneous platforms and interfaces. Therefore, this would increase the cost of design and maintenance of a central control system.

The proposed solution for the trigger system of the CMS experiment, presented in this document, copes with these three complexity dimensions, and can be an example for other experiments.

1.2 Document purpose

This document is intended to specify the functional and non-functional requirements of the Trigger Supervisor, its design and operational details, and the components that will be delivered in order to facilitate a smooth integration of the trigger software in the context of the RCMS and the standalone operation of the trigger. It complements the integration of Run Control and the Detector Control System and serves as a reference document endorsed by the CMS Trigger and DAQ groups.

1.3 Document overview

The document is divided into the following chapters: chapter 1 contains the introduction. Chapter 2 sets the functional and non-functional requirements for the Trigger Supervisor, chapter 3 presents the design, and chapter 4 describes the components of the package that will be delivered to the trigger subsystems and a methodology of work associated with the TS project. Chapter 5 closes the document with a summary and the conclusions. Acronyms are listed after the last chapter.

1.4 Definitions

Authentication

Authentication is the process of verifying the identification of a user/entity. This is necessary to protect against unauthorized access to a system or to the information it contains. Typically, authentication takes place using a password.

Authorization

Authorization is the process of deciding if a requesting user/entity is allowed to have access to a system service.

Configuration management

Configuration management is the discipline applying technical and administrative controls to: 1) identification and documentation of physical and functional characteristics of configuration items, 2) any changes to characteristics of those configuration items, 3) recording and reporting of change processing and implementation of the system. The term 'software configuration management' will be used in the Trigger Supervisor context in the following, in order to distinguish it from the detector and trigger hardware detector configuration management.

Identification

Identification includes the processes and procedures employed to establish a unique user/entity identity within a system.

Subsystem

The term subsystem is used to refer both to the trigger subsystems and to the subdetectors. For instance, interconnection and self test operations involve not only the trigger subsystems but also the subdetectors themselves (section 3.2.3).

Trigger subsystems

The CMS Level-1 Trigger is described in [2]. The trigger subsystems relevant for the Trigger Supervisor are: Regional Calorimeter Trigger, Global Calorimeter Trigger, RPC (Resistive Plate Chamber) Trigger, CSC (Cathode Strip Chamber) Track Finder, DT (Drift Tube) Track Finder, Global Muon Trigger, Global Trigger and Trigger Control System (TCS). The Local Triggers or Trigger Primitive Generators (TPG) for calorimetry and muons, at the bottom end of the trigger chain, are based on energy deposits in calorimeter trigger towers and track segments in muon chambers, respectively. Their configuration is not performed through the TS, but is under the responsibility of the corresponding subdetector. Nevertheless, they may participate in certain interconnection tests (section 3.2.3.2). The Global Trigger takes the decision to reject or to accept an event for further processing by the HLT. It is based on input from the Global Calorimeter and Global Muon Triggers, which process information from the Regional Triggers of the calorimeters and the muon system. The CMS muon trigger is special in the sense that

both the tracking chambers, made up of DT chambers in the central part and CSC chambers in the endcaps, and the dedicated RPC trigger chambers take part in it.

TTC/TTS partition

Different subsystems of CMS may be operated independently concerning the trigger. For this purpose, the experiment is subdivided or partitioned into 32 so called TTC/TTS partitions [8]. Each TTC/TTS partition represents a major component of a subsystem.

TCS partition

This is a group of TTC/TTS partitions controlled by the Trigger Control System (TCS) [8]. During normal physics data taking there is only one single TCS partition. For commissioning and testing up to 8 TCS partitions are available, which each receive their own Level-1 Accept signals.

2 Requirements

This chapter discusses the main requirements of the TS. These have been separated in those related to the system functionality and the rest. The requirements are outlined in the following sections.

2.1 Functional requirements

The TS is conceived to be a central node that offers a high level API to facilitate the setting of a concrete configuration of the trigger system, to launch test that involve several subsystems or to monitor a number of parameters in order to check the correct functionality of the trigger system. In addition, the TS will provide an access point to the online software infrastructure of each trigger subsystem. The TS framework is being designed as an open framework capable to adopt new functionalities required by specific subsystems.

2.1.1 Configuration

The first functionality offered by the TS will be the configuration of the trigger system. This functionality will hide to the controller the complexity of operating the different trigger subsystems in order to set up a given configuration.

2.1.2 HLT Synchronization

In order to properly configure the HLT, it will be necessary to provide a mechanism to synchronize the propagation of the trigger system configuration to the HLT. This synchronization will also be necessary for the HLT to verify the Level-1 Trigger decision by properly configured ORCA simulations of the trigger boards.

2.1.3 Test

The TS will offer an interface to test the trigger system. Two different test services will be provided: the self test, intended to check each trigger subsystem individually; and the interconnection test service, intended to check the connection among subsystems.

2.1.4 Monitoring

The TS interface will facilitate to monitor the necessary information that assures the correct functionality of the trigger subsystems, such as the ORCA simulations of the trigger hardware running on the HLT, subsystem specific monitoring data (e.g., spy memories), and the necessary information for synchronization purposes (e.g., histograms of occupancy per bunch crossing number and the correlation function).

2.1.5 User management

During the experiment commissioning, the different subdetectors will be tested independently, and many of them might be tested in parallel. In other words, several RC sessions, running concurrently, will need to access the trigger system. Therefore, it is necessary that the TS coordinate the access to the common resources (e.g., the TCS, the trigger subsystems and the configuration DB). In addition, it is necessary to control the access to the trigger system in order to determine which users/entities (controllers) can have access to it and what privileges they have. A complete access control protocol will be defined that will include: the identification, authentication, and authorization processes.

In order to create a policy to grant access to the shared resources first a hierarchy of users needs to be defined. The simplest one is the following:

- Super user: This user is allowed to set configuration changes with global effect. This means that he can fully configure the TCS, any trigger subsystem and can modify the configuration of any TCS partition. He can also run self and interconnection tests.
- Partition manager: This user is allowed to set configuration changes whose effect is strictly restricted to a given TCS partition. He can also run self and interconnection tests.
- Monitoring user: This user can just monitor.

2.1.5.2 User privileges and responsibilities

In this section it is explained how the different privileges/responsibilities are distributed among the trigger coordinator, the TS system, the super user, partition managers, and monitoring users.

Trigger coordinator privileges/responsibilities (this person has full access to the user DB):

- To maintain the list of users in the user DB (section 2.1.5.3). This responsibility can be shared with already existing super users.

Super user privileges:

- To launch self test operations in all trigger subsystems.
- To launch interconnection tests among subsystems.
- To launch monitoring operations.
- To assign a time slot to each TCS partition (global effect).
- To configure the trigger throttle logic (global effect).
- To configure the default BC table for every TCS partition (global effect).
- To configure the common parameters at the top of the level-1 hierarchy, in the Global Trigger Logic (GTL) [9]: Algorithms and thresholds (global effect).
- To configure the trigger subsystems (global effect). This configuration includes all trigger subsystems. The configuration of the TPG modules is under the responsibility of each subdetector.

Partition manager privileges:

- To launch self test operations in all trigger subsystems.
- To launch interconnection tests among subsystems.

- To launch monitoring operations.
- To create a new TCS partition (To assign a set of TTC/TTS partitions to a TCS partition) (local effect).
- To configure the corresponding parameters of the GTL board that will just affect the corresponding L1A signal assigned to the TCS partition (local effect). (The TS is responsible to check that reconfiguration in one TCS partition does not affect other TCS partitions).
- To assign a set of algorithms to the corresponding L1A delivered by the Final Decision Logic (FDL module) of the Global Trigger [10] (local effect).
- To modify the BC table of a given TCS partition (local effect).
- To start/stop the delivery of L1A signals for a given TCS partition.

Monitoring user privileges:

- To launch monitoring operations.

A detailed explanation of how all those rules are controlled can be found in section 3.1, where the design of the TS is presented (e.g., configuration control details described in point 2 of the configuration service in section 3.2.1).

2.1.5.3 User DB

The configuration DB will also maintain the list of users with the corresponding level of access as well as the necessary information to authenticate them. The list of users can be administrated by any super user, and the list of super users will be set up by the trigger coordinator.

The user DB will contain the following information for every user:

- Login
- Password
- Type of user (e.g., super user, partition manager and monitoring user)
- Partitions that can be managed: this is a 32 bit binary number, where each bit corresponds to a TTC/TTS partition. If the user is allowed to be the partition manager of a TCS partition that contains the TTC/TTS partition number n ($0 \leq n \leq 31$) then the n^{th} position of this binary number is 1, otherwise it is 0.

2.1.5.4 Authorization policies

The policy to authorize a user is as follows:

- A user can be authorized as a super user if this user is allowed to be super user (the trigger coordinator is in charge to administrate the list of super users in the user DB).
- A user can be authorized as partition manager of a given TCS partition if all the TTC/TTS partitions that constitute the TCS partition can be managed by that user (section 2.1.5.3) and none of these TTC/TTS partitions are managed by another user.
- All existing users in the user DB can be authorized as monitoring users.

2.1.5.5 User management tasks

The Trigger Supervisor is responsible for the following user management tasks:

- To identify, authenticate and provide an authorization protocol.

- To check that a given user, with a given level of rights, can perform a given operation.
- To check that an authorized action is not contradictory to a running operation.
- To verify that no more than 1 super user is connected.
- To coordinate the access of different users to common resources (e.g., configuration DB, trigger subsystems). For instance, the TS is responsible to check that the reconfiguration in a concrete TCS partition, launched by a partition manager, does not affect other TCS partitions.

2.1.6 Automatic start up

In order to maximize subsystem independence and client decoupling (section 2.2.3), a hierarchical start up mechanism has been chosen (section 3.2.5 describes the operational details). As will be described later, the TS is organized in a tree like structure, with a central node and several leafs. The first RC session or controller will be responsible to start up the TS central node, and in turn this will offer an API that allows to start up the TS leafs and the online software infrastructure of the corresponding trigger subsystem.

2.1.7 Logging support

The TS will provide logging mechanisms in order to support the users to carry out troubleshooting activities in the event of problems. Logbook entries should be time-stamped and should include all necessary information such as the details of the action and the identity of the user responsible. The log registry is available online and is also recorded for off-line use.

2.1.8 Error handling

An error management scheme, compatible with the global error management architecture, will be necessary. This must provide a standard error format, and remote error handling and notification mechanisms.

2.1.9 User support

2.1.9.1 User display

A GUI will be provided. This will allow a standalone operation of the TS and will help the user to interact with the TS framework, and to visualize the state of a given operation, and monitoring data. From the main GUI it will be possible to open specific GUIs for each trigger subsystem. Those will be based on a common skeleton that will be fulfilled by the trigger subsystem developers following a given methodology described in a document that will be provided.

2.1.9.2 User assistance

An adequate online help facility shall be provided to help the user operate the TS, since many of the users of the TS will not be experienced and may not have received detailed training.

2.2 Non-functional requirements

2.2.1 Low level infrastructure independence

The design of the TS should be as much as possible independent of the online software infrastructure (OSWI) of any subsystem. In other words, the OSWI of a concrete subsystem should not drive any important decision in the design of the TS. This requirement is intended to minimize the TS redesign due to the evolution of the OSWI of any subsystem.

2.2.2 Subsystem control

The TS should offer the possibility to operate a concrete trigger subsystem. Therefore, the design should be able provide at the same time a mechanism to coordinate the operation of a number of trigger subsystems, and a mechanism to control a single trigger subsystem.

2.2.3 Controller decoupling

The TS must operate in different environments: inside the context of the common experiment operation, but also independently of the other CMS subsystems, such as, during the phases of commissioning and maintenance of the experiment, or during the trigger subsystem integration tests. Due to the diversity of operational contexts, it will be useful to facilitate the access to the TS through different technologies: RCMS, java applications, web browser or even batch scripts. In order to allow such heterogeneity of controllers, the TS design must be totally decoupled from the controller, and the following requirements should be taken into account:

- The logic of the TS should not be split between a concrete controller and the TS itself.
- The technology choice to develop the TS should not depend on the software frameworks used to develop a concrete controller.

In addition, the logic and technologic decoupling from the controller will increase the maintenance and the evolution potential of the TS, it will also increase development and debug options, and will reduce the complexity of operating the trigger system in a standalone way.

2.2.4 Multi user

During the commissioning phase, several RC sessions will be running concurrently. Each of them will be responsible to operate a different TCS partition. In addition, the TS should allow standalone operations (not involving the RC framework), for instance, to execute tests or monitor the trigger system. Therefore, it will be necessary to facilitate that several clients can be served in parallel by the TS.

2.2.5 Interface

2.2.5.1 XML-based

In order to facilitate the integration, the interface description and implementation between the controller and the TS follows a web service based approach [11]. The chosen communication protocol to send commands and state notifications is SOAP (Simple Object Access Protocol) [12], and the representation format of exchanged data is XML (Extensible Markup Language) [13]. Inside the CMS collaboration the same scheme has also been chosen by XDAQ, DCS (PVSS SOAP Interface, PSI) and Run Control (Function Managers). different operating systems

The format of the transmitted data and the SOAP messages is specified using the XML schema language [14], and the WSDL language [15] is used to specify the location of the services and the methods the service exposes. On the other hand, XML has one big drawback: XML uses by default textual data representation, which causes much more network traffic to transfer data. Even BASE64 or Uuencoded byte arrays are approximately 1.5 times larger than a binary format. Furthermore, additional processing time is required for translating between XML and native data representations. Therefore, the current approach is not well suited for devices generating abundant amount of real-time data, but is still valid for configuration, monitoring, and slow control purposes. Should additional performance be required, this drawback can be overcome by using a binary serialization package provided within the CMS online software project and I2O messaging [16]. In this case, the WSDL could not be used to specify the I2O message format.

2.2.5.2 Asynchronous protocol

Due to the long time required to finish the execution of configuration and test commands, an asynchronous protocol will be necessary to interface the TS. This means that the receiver of the command will reply immediately

acknowledging the reception, and that this receiver will send another message to the sender once the command is executed. An asynchronous protocol may improve the usability of the system because the human computer interface will not block until the completion of the requested operation.

2.2.5.3 Private network access

The OSWI of the trigger subsystems will likely be installed inside a private network. Therefore, it will be necessary to provide a way to overcome this border.

3 Design

This chapter presents a design for the TS that fits the functional and non-functional requirements listed in chapter 2. The architecture is composed of a central node in charge to coordinate the access to the different subsystems, namely the trigger subsystems and subsystems concerned with the interconnection test service (section 3.2.3.2), and a customizable TS leaf (section 4.2) for each of them that will offer the central node a well defined interface to operate the online software infrastructure of each subsystem. Fig. 1 shows the architecture of the TS.

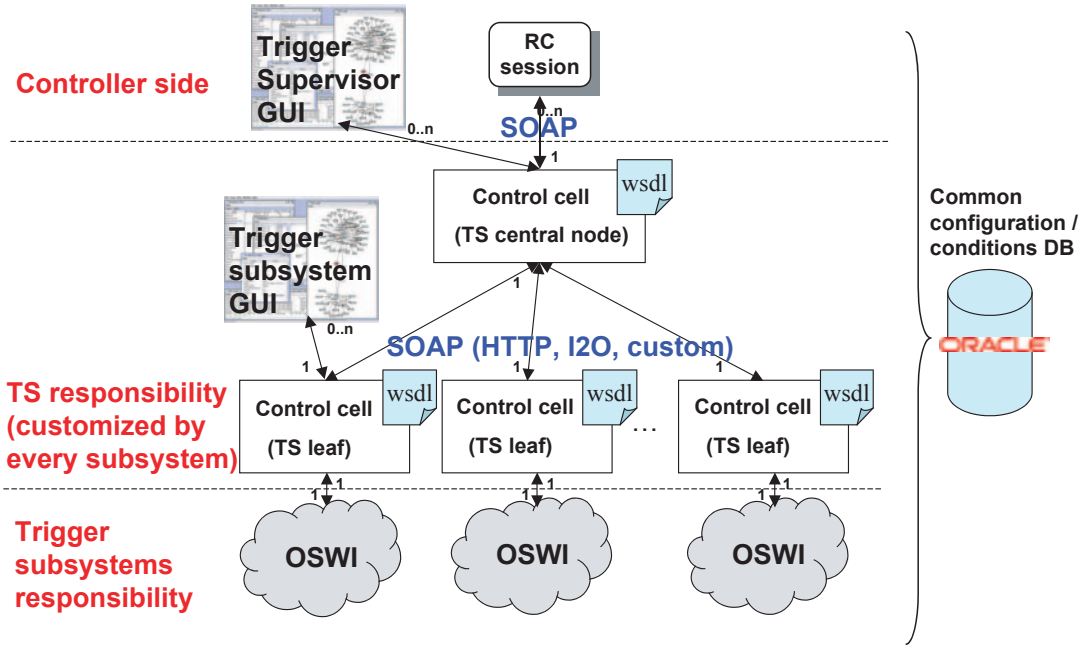


Figure 1: Architecture of the Trigger Supervisor.

Each node of the TS can be accessed independently, fulfilling the requirement outlined in section 2.2.2. The available interfaces and location for each of those nodes are defined in a WSDL document. Both the central node and the TS leafs are based on a single common building block, the *control cell*. Each subsystem group will be responsible to customize a control cell, and to keep the consistency of the available interface with the interface described in the corresponding WSDL file.

The presented design is not driven by the available interface of the OSWI of a concrete subsystem (section 2.2.1). Therefore, this will improve the evolution potential of the low level infrastructure and the TS. Moreover, the design of the TS is logically and technologically decoupled from any controller (section 2.2.3). In addition, the distributed nature of the TS design will facilitate a clear separation of responsibilities and a distributed development.

It is interesting to note that the usage of the common control cell software framework could be used in a variety of different control network topologies (e.g., N-level tree or peer to peer graph).

CMS has developed a C++ based data acquisition framework [17, 18] called XDAQ. The OSWI of all subsystems is based on this distributed programming framework. Therefore, an obvious option is to develop the TS using this framework. The following reasons justify this choice:

- The software frameworks used in both the TS and the subsystems are homogenous.
- For a faster messaging protocol I2O messages could be used instead of being limited to messages according to the SOAP communication protocol.
- Monitoring and security packages are available.
- XDAQ development is practically finished, and its API is considered already stable. The launch of the final production version is imminent.

3.1 The control cell

The architecture of the TS is characterized by its tree topology, where all tree nodes are based on a common building block, the control cell. Fig. 2 shows the architecture of the control cell.

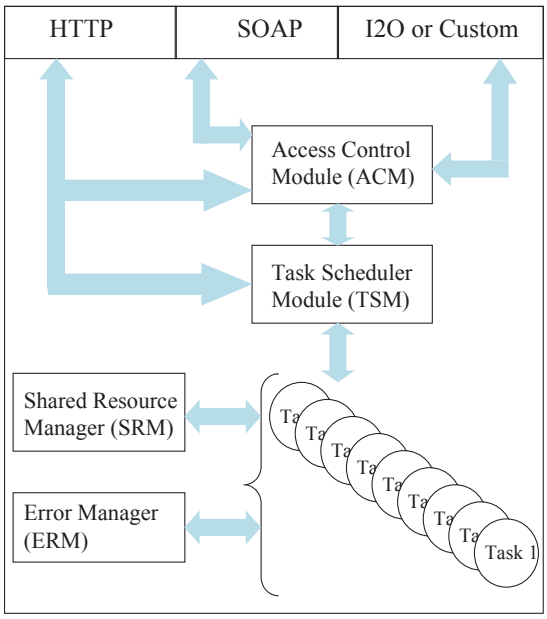


Figure 2: Architecture of the control cell.

The basic building block of the TS, the control cell, is a program that offers the necessary functionalities to coordinate the control operations over other software systems, for instance the OSWI of a concrete trigger subsystem, an information server, or even another control cell.

Each cell can work independently of the rest (fulfilling the requirement of section 2.2.2), or inside a more complex topology. The following points describe the components of the control cell:

- Control Cell Interface (CCI): This is the external interface of the control cell. Different protocols will be available. An HTTP interface will be provided using the XDAQ facilities; this will facilitate a first entry point from any web browser. A second interface based on SOAP will also be provided in order to ease the integration of the TS with the Run Control or any other controller that requires a web service interface. Future interface extensions are foreseen. For instance, if performance is required an I2O interface will be implemented. Each control cell will have an associated WSDL document that will describe its interface. The

information contained in that document instructs any user/entity how to properly operate with the control cell.

- Access Control Module (ACM): Each module is responsible to identify and authenticate every user or entity (controller) attempting to access, and to provide an authorization protocol. The access control module will have access to a user list, which will provide the necessary information to identify and authenticate, and the privileges assigned to each controller. Those privileges are used to check whether or not an authenticated controller is allowed to execute a given operation.

- Task Scheduler Module (TSM): This is the most complex module of the control cell. This module is in charge to manage the command requests and to forward the answer messages. The basic idea is that a set of available operations exist that can be accessed by a given controller. Each operation corresponds to a finite state machine (FSM). The default set of operations is customizable and extensible. The TSM is also responsible to avoid launching operations that could enter into conflict with other running operations (e.g., simultaneous self test operations within the same trigger subsystem, interconnection test operations that cannot be parallelized). The extension and/or customization of the default set of operation could change the available interface of the control cell. In this case, the corresponding WSDL should be updated.

- Shared Resources Manager (SRM): This module is in charge to coordinate the access to shared resources (e.g., the configuration DB, other control cells, or a trigger subsystem online software infrastructure). Independent locking services for each resource are provided.

- Error Manager (ERM): This module will provide the error management of all errors not solved locally, which have been generated in the context of the control cell, and also the management of those errors that could not be resolved in a control cell immediately controlled by this one. It is important to note that both the error format and the remote error notification mechanism will be based on the global CMS distributed error handling scheme.

It is interesting to note that the control over what operations can be finally executed is distributed among the ACM (user access level control, e.g., a user with monitoring privileges can not launch a self test operation), the TSM (conflictive operation control, e.g., to avoid running in parallel operations that could disturb each other), and inside the commands code of each operation (e.g., to check that a given user can set up a configuration identified with a given 'key'. More details are given in section 3.2.1).

3.2 Trigger Supervisor services

The Trigger Supervisor services are the final functionalities offered by the Trigger Supervisor. These services emerge from the collaboration of several nodes of the TS tree. In general, the central node will always be involved in all services coordinating the operation of the necessary TS leafs (section 2.1.1). The goal of this section is to describe, for each different service, what the default operations are in both the central node of the TS and in the TS leafs, and how the services emerge from the collaboration of these distributed operations. It should be noted that a control cell operation is always a FSM.

3.2.1 Configuration

This service is intended to facilitate the hardware configuration of the trigger system, which includes the setting of registers or look-up tables and downloading the trigger logic into the FPGA's of the electronics boards. The configuration service requires the collaboration of the central node of the TS and all the TS leafs. Each control cell involved will implement the operation represented in Fig. 3.

Due to the asynchronous interface, it is also necessary to define transition states such as 'Configuring' and 'Enabling', which indicate that a transition is in progress. All commands are executed whilst the FSM is in a transition state. Therefore, the error state can be invoked from a transition state. Fig. 4 shows how the different nodes of the Trigger Supervisor collaborate in order to fully configure the trigger system.

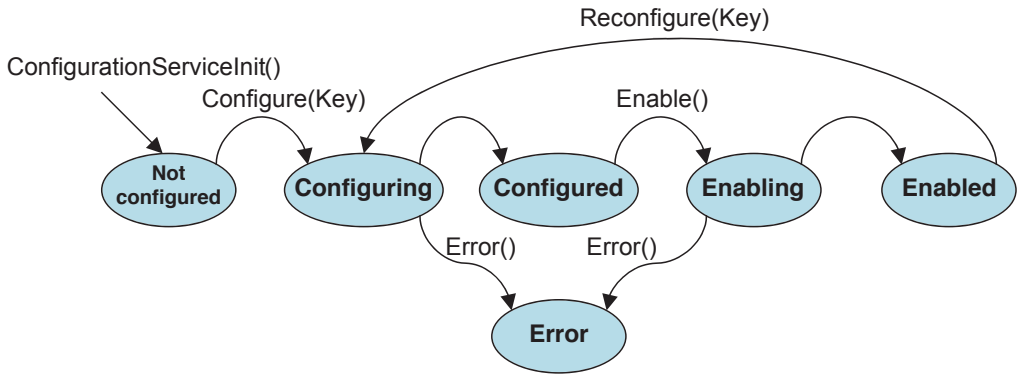


Figure 3: Configuration operation.

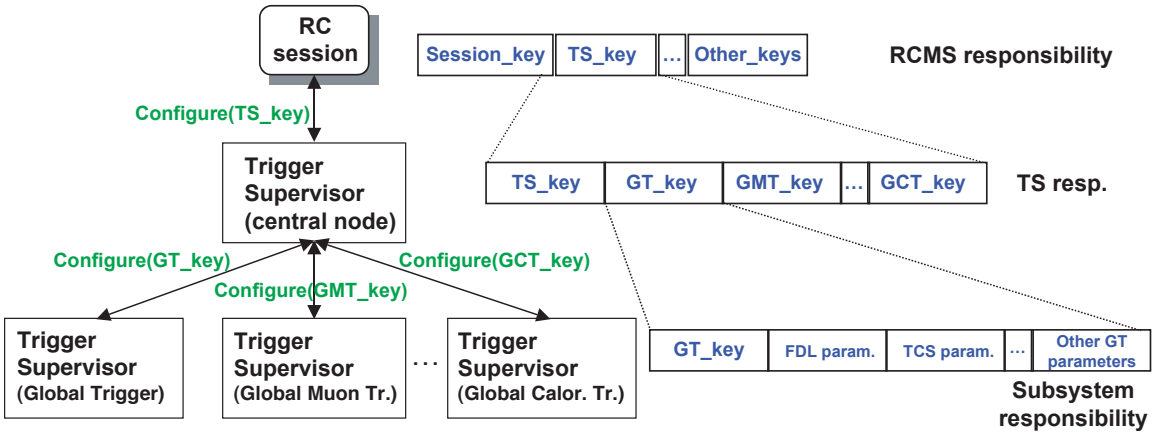


Figure 4: Configuration service.

This is the sequence of steps that a controller of the TS should follow in order to properly use the configuration service:

- Send a `'ConfigurationServiceInit()'` command to the central node of the TS.
- Once the operation reaches the 'not configured' state, the next step is to send a `'Configure(TS_key)'` command, where 'TS_key' identifies a set of 'trigger_key's, one per trigger subsystem that is to be configured. The `'Configure(TS_key)'` command is in charge to initiate the configuration operation in the relevant TS leafs. The configure command in the configuration operation of each TS leaf will check whether or not the user is allowed to set such a configuration identified with a given 'trigger_key'. This means that each trigger subsystem has the full control over who and what can be configured. This means also that the list of users in the central node of the TS will also be replicated in the TS leafs.
- Once the configuration operation of the TS leafs reaches the 'Configured' state, the configuration operation in the central node of the TS jumps also to the 'Configured' state.
- Send an `'Enable()'` command. This fourth step is just a switch-on operation.

The configuration operation represents the configuration state of the trigger system. Once the configuration op-

eration reaches the ‘Enabled’ state, everything is ready, from the point of view of the trigger system, to run the experiment.

Each trigger subsystems group will have the responsibility to customize the configuration operation of its own control cell and thus will have to implement the commands of the FSM. The central node of the Trigger Supervisor owns the data that relates a given trigger key to the trigger subsystem keys.

The presented configuration service is flexible enough to allow a full or a partial configuration of the trigger system. In the second case, the ‘TS_key’ identifies just a subset of ‘trigger_key’s, one per trigger subsystem that is to be configured, and/or each ‘trigger_key’ identifies just a subset of all the parameters that can be configured for a given trigger subsystem. As will be shown in section 3.4, the configuration DB is common to both the central node of the TS and the TS leafs. Each trigger subsystem will be responsible to populate the configuration DB and to assign key identifiers to sets of configuration parameters. The exact methodology is still under development.

3.2.2 HLT interface

This section has to be read as a complement of section 3.2.1. Any reconfiguration of the trigger system should be synchronized with the propagation of the new configuration table to the filter farm, as it was required in section 2.1.2. The following steps show how a controller of the TS should behave in order to properly reconfigure the trigger system using the configuration service:

1. Once the trigger system is configured, the configuration operation in the central node of the TS will be in the ‘Enabled’ state.
2. Send a `‘Reconfigure(key)’` command. The following steps show how this command behaves:
   (a) Stop the generation of L1A signals.
   (b) Follow the steps from 2 and 3 of section 3.2.1 and
   (c) Jump to the state ‘Configured’.
3. The controller is also responsible to propagate the configuration changes to the filter farm hosts in charge of the HLT and the ORCA simulation through the configuration/condition DB.
4. Send an `‘Enable()’` command: This signal will be sent by the controller to confirm the propagation of configuration changes to the filter farm hosts in charge of the HLT and the ORCA simulation. This command will be in charge to resume the generation of L1A signals.

Therefore, Run Control is in charge to coordinate the configuration of the TS and of the HLT. There is no special interface between the central node of the TS and the HLT.

3.2.3 Testing

The TS will offer two different test services: the self test service and the interconnection test service. The following sections describe both.

3.2.3.1 Self test

This service is intended to check that each individual subsystem is able to operate as foreseen. If anything fails during the test of a given subsystem, an error report is returned, which can be used to define the necessary corrective actions. The self test service can involve one or more subsystems. In the second, more complex case, the self test service requires the collaboration of the central node of the TS and all the corresponding TS leafs. Each control cell involved will implement the same self test operation. The self test operation running in each control cell is a FSM with just two states: ‘halt’ and ‘tested’. This is the sequence of steps that a controller of the TS should follow in order to properly use the self test service:

1. Send `‘SelfTestServiceInit()’` command. Once the self test operation is initiated, the operation reaches the ‘halt’ state (initial state).

2. Send a `'RunTest(LogLevel)'` command, where the parameter 'LogLevel' specifies the level of detail of the error report. An additional parameter 'type', in the `'RunTest'` command, might be used to distinguish among different types of self test.

The behavior of the `'RunTest'` command depends on whether it is the self test operation of the central node of the TS, or a self test operation in a TS leaf. In the central node of the TS, the `'RunTest'` command is in charge to follow the above sequence for each TS leaf, and collect all error reports coming from the TS leafs. In the case of the TS leaf, the `'RunTest'` command will implement the test itself and will generate an error report that will be forwarded to the central node of the TS. It is important to note that the error report will be generated in a standard format specified in a XML Schema Document (XSD document).

3.2.3.2 Interconnection test

This service is intended to check the connections among subsystems. In each test, several trigger subsystems and subdetectors can participate as Sender/s or Receiver/s. Fig. 5 shows a typical scenario and the participants involved in an interconnection test.

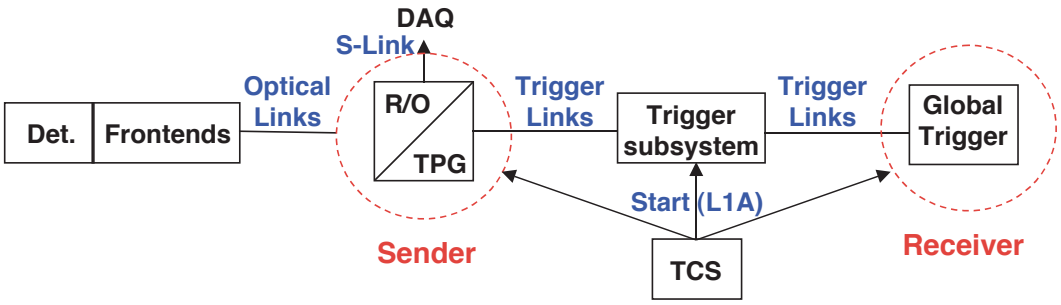


Figure 5: Typical scenario of an interconnection test.

The interconnection test service requires the collaboration of the central node of the TS and some of the TS leafs. Each control cell involved will implement the operation represented in Fig. 6.

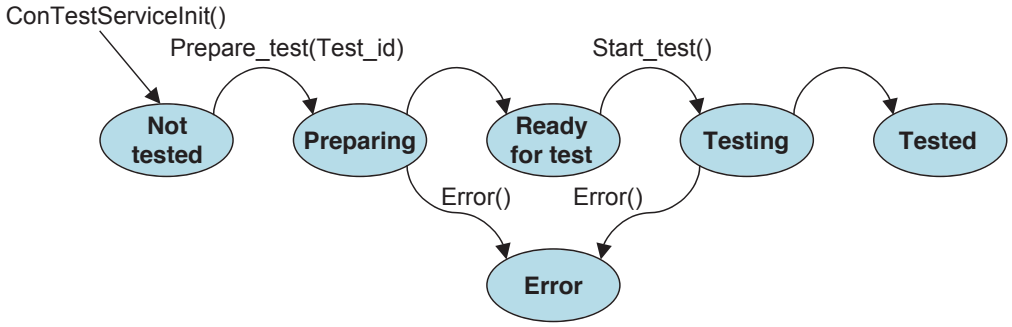


Figure 6: Interconnection test operation.

This is the sequence of steps that a controller of the TS should follow in order to properly use the interconnection test service:

1. Send a `'ConTestServiceInit()'` command.
2. Once the operation reaches the 'Not tested' state, the next step is to send a `'PrepareTest(Test_id)'`. This command implemented in the central node of the TS will do the following steps:

(a) Retrieve from the configuration DB the necessary information concerning the interconnection test whose identifier is 'Test_id'. It will also be necessary that the TS identify itself in order to retrieve just the information that concerns it (e.g., {#test_id, sender_id/s, receiver_id/s, coordination data}).

(b) To send a `'ConTestServiceInit()'` command to sender/s and receiver/s.

(c) Send `'Prepare_test()'` command to sender/s and receiver/s.

(d) Wait for 'Ready_for_test' signal from all sender/receiver.

3. Once the operation reaches the 'Ready' state, the next step is to send a `'Start_test()'` command.

4. Wait for results.

This is the sequence of steps that the TS leafs acting as senders/receivers should follow when they receive the `'PrepareTest(Test_id)'` command from the central node of the TS:

1. Retrieve from the configuration DB the necessary information concerning the interconnection test whose identifier is 'Test_id'. It will also be necessary that the TS leaf identify itself in order to retrieve only the information that concerns it (e.g., which role: sender or receiver, test vectors to be sent or to be expected).

2. Send a 'Ready_for_test' signal to the central node of the TS.

3. Wait for `'Start_test()'` command.

4. To do the test, and generate the test report to be forwarded to the central node of the TS (if the TS leaf is a receiver).

Like in the self test case, it is important to note that the error report will be generated in a standard format specified in a XSD document.

In this case, compared to the configuration service, the central node of the TS can already check whether or not a given user can launch interconnection test operations. However, the TSM of each TS leaf will still be in charge to check whether or not acting as a sender/receiver is in conflict with an already running operation.

Each subdetector should also customize a control cell in order to facilitate the execution of interconnection tests that involve the TPG modules.

3.2.4 Monitoring

The monitoring service will be implemented by an operation running in a concrete TS leaf or as a collaborative service where an operation, running in the central node of the TS, is monitoring the monitoring operations running in a number of TS leafs.

The basic monitoring operation is a FSM with just two states: 'monitoring' and 'stop'. Once the monitoring operation is initiated, the monitoring process is started. At this point, any controller can retrieve items by sending `'pull'` commands. A more advanced monitoring infrastructure will be offered in a second development phase where a given controller will receive monitoring updates following a `'push'` approach. This second approach would facilitate the implementation of an alarm mechanism.

The proposed monitoring scheme depends on the monitoring infrastructure provided by XDAQ. Therefore, both consumer and producers must be implemented with this framework.

3.2.5 Start up

From the point of view of a controller (RC session or standalone client), the whole trigger system is one single resource, which can be started sending three commands. Fig. 7 shows how this process is carried out. This approach will simplify the implementation of the client.

The first client that wishes to operate with the TS must follow the following steps:

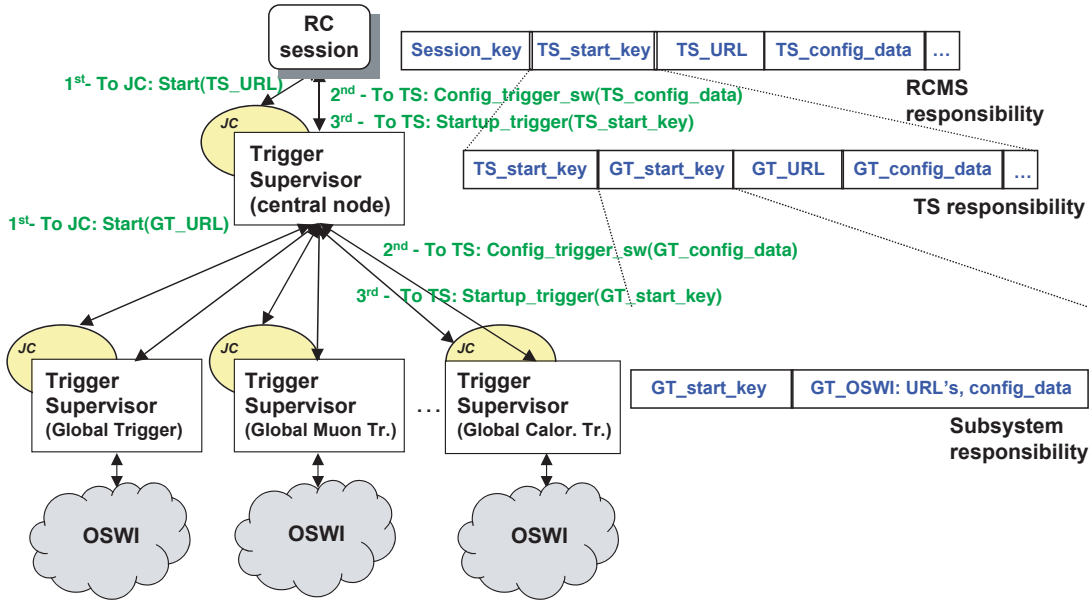


Figure 7: Start up service.

1. Send a `'Start(TS_URL)'` command to the job control (JC) daemon in charge to start up the central node of the TS, where 'TS_URL' identifies the URL from where the compiled central node of the TS can be retrieved.

2. Send a `'Config_trigger_sw(TS_config_data)'` command to the central node of the TS in order to properly configure it. Steps 1 and 2 are separated to facilitate an incremental configuration process.

3. To send a `'Startup_trigger(TS_start_key)'` command to the central node of the TS. This command will send to each TS leaf the same sequence of three commands, but now the command parameters are retrieved from the configuration DB register identified with the 'TS_start_key' index.
   The `'Startup_trigger(TSLeaf_start_key)'` command that receives the TS leaf is in charge to start up the corresponding online software infrastructure.

The release of the TS nodes is also hierarchic. Each node of the TS (i.e., TS central node and TS leafs) will maintain a counter of the number of controllers that are operating on it. When a controller wishes to stop operating a given TS node, it has to demand the value of the reference counter from the TS node. If it is equal to 1, then the controller will send a `'Release_node'` command and will wait for the answer. When a TS node receives a `'Release_node'` command this will behave like the controller outlined above in order to release the unnecessary software infrastructure.

3.3 The GUI

Together with the basic building block of the TS or control cell, an interactive graphic environment to interact with it will be provided. It will feature a display to help the user/developer to operate the control cell, and will cope with the requirement outlined in section 2.1.9.1. Two different interfaces will be provided:

- HTML: The control cell will provide an HTTP interface that will allow to fully operate the control cell and to visualize the state of any running operation. The HTTP interface will provide an additional entry point to the control cell (section 3.1), bypassing the ACM, in order to offer a larger flexibility in the development and debug phases.

- Java: A generic controller developed in java will provide to the user an interactive window to operate the control cell through a SOAP interface. This java application will also be an example of how to interact with the monitoring operations offered by the control cell, and graphically represent the monitored items. This java controller can be used in the future by Run Control as an example of how to interact with the TS.

3.4 Configuration/conditions DB

A common configuration/conditions DB approach, as it is shown in Fig. 1, will be used by all the trigger subsystems and the TS. The DB infrastructure will be provided partially by each trigger subsystem and the CMS Data Base Working Group (DBWG). The DBWG will provide the following components:

- HW infrastructure: servers.
- SW infrastructure: likely based on Oracle, scripts and generic GUIs to populate the data bases, methodology to create customized GUIs to populate subsystem specific configuration data.

Each trigger subsystem should provide:

- Specific DB structures, which will be integrated in the CMS configuration DB once the necessary infrastructure is in place:
  - Configuration data and mechanisms to relate key identifiers to sets of configuration parameters (section 3.2.1).
  - Access control information (section 2.1.5.3).
  - Basic information of interconnection tests (section 3.2.3.2).
  - Sender/Receiver interconnection test tasks (section 3.2.3.2).
- Custom GUIs to populate the DB (based on DBWG recommendations).

4 Deliverables and configuration management

The development of the TS will require the distribution of its implementation among the trigger subsystems and the TS groups. The skeleton of the basic control cell, presented in section 3.1 will be delivered to every trigger subsystem group. Each subsystem will be responsible to customize it. The necessary documentation will be written in order to ease this process, and a set of configuration management actions will be proposed in order to improve the communication of the system evolution and the coordination among trigger subsystem development groups.

4.1 Skeleton

The following software packages will be provided to each trigger subsystem:

- Generic control cell: common SW skeleton to be customized by every subsystem (section 3.1).
- Pluggable set of default operations: FSMs for configuration, monitoring and testing and proposed API of commands to be filled by subsystems (section 3.2).
- Skeleton of a customizable GUI (section 3.3)

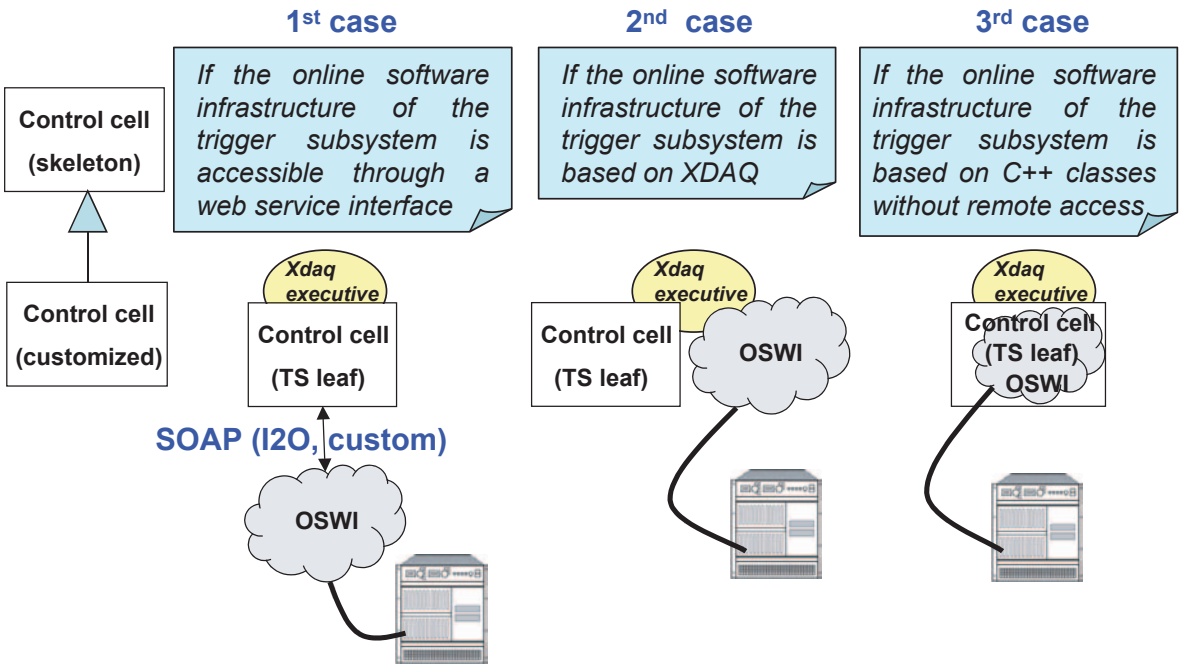


Figure 8: Integration scenarios.

4.2 Development methodology

A template TS leaf is provided for each subsystem. Each individual subsystem will however be responsible to customize its own leaf. Fig. 8 shows the different scenarios for the customization of the TS leaf, as a function of the available online software infrastructure. The OSWI is the software system running locally on the corresponding trigger subsystem server. This contains the driver that facilitates the access to the VME crate.

Scenarios

- Web service access to the OSWI: in this case the customization of the control cell will use the web service API to access to the hardware.
- XDAQ-based OSWI: in this case one can either use the SOAP API to access to the XDAQ services (first case) or to bind the control cell and the OSWI to the same XDAQ executive. In this second case, it is possible to use the XDAQ infospace to communicate the control cell and the OSWI.
- C++ API: in this third case the available OSWI is a set of classes that facilitates the access to the hardware. The control cell must run locally in the corresponding trigger subsystem server.

In any of the three scenarios, the SOAP interface of the corresponding TS leaf can change. Each trigger subsystem will be responsible to keep a consistent WSDL file that properly specifies this API.

The customization flexibility of the control cell facilitates the evolution of the underlying OSWI and the integration of several different interfaces (i.e., SOAP, I2O, C++), whilst keeping a well defined interface with the central node of the TS. In order to ease the integration process, the following documentation will also be delivered:

- Global Trigger integration note.
- How to customize the operations and to keep the consistency of the WSDL file.
- How to customize the TSM in order to control the execution of contradictory operations.

- How to interact with the ACM.
- How to interact with the configuration/conditions DB.
- How to interact with the monitoring services.
- How to customize the GUI in order to set histograms, and graphic information of monitored data.
- How to deal with the error manager (ERM) compatible with the global error management architecture.
- How to define interconnection tests and interact with the configuration DB.

4.3 Software configuration management

The online software infrastructure of the trigger system will evolve during the development phase, but also during the installation, the commissioning and the operational phases. An accurate and centralized control over the evolution of the system configuration has the following advantages:

- To facilitate the exact replication of the trigger system.
- To improve the communication of the software evolution.
- To facilitate the coordination among trigger subsystem groups.
- To ease the resource sharing among trigger subsystem groups.

An exhaustive software configuration management (section 1.4) would require tracking several parameters. On the other hand, a more suitable and realistic approach is to adopt different actions in a progressive way. Two initial actions are proposed. Those will be especially useful in the development of the TS because the successful completion of this project will require a distributed development and suitable collaboration procedures.

4.3.1 Initial actions

As a part of a strategy for a proper configuration management of the trigger online software infrastructure two initial actions are proposed: 1) Common CVS repository and 2) the adoption of a generic Makefile.

Common CVS repository

A common CVS repository for all the online software infrastructure of the trigger has been created. This will facilitate the production of trigger software releases, which in turn would simplify the deployment of the whole trigger online software infrastructure. In addition, the extensive usage of a common CVS repository will ease the sharing of software components among subsystem groups, and will improve the communication and coordination among them.

This coordination will be very important in the development of the TS. The trigger repository will be the way how the trigger subsystem groups will communicate the evolution of their TS leaf interface. Every change in a TS leaf interface must be consistently documented in the corresponding WSDL file, and this file must be immediately checked in the trigger repository.

Generic Makefile

The set up of a common repository comes together with a proposal to homogenize the build process of software modules. An extensive usage of this methodology would homogenize the build process of the trigger software that would facilitate a more automatic deployment of the trigger online software infrastructure, and would prepare this infrastructure to be integrated with the online software infrastructure of the DAQ.

5 Summary and Conclusions

The requirements for the Trigger Supervisor have been identified and a suitable architecture has been proposed and approved by the Trigger and DAQ groups. This is composed of a central node in charge to coordinate the access to the different subsystems, and a customizable TS leaf for each of them that will offer to the central node a well defined interface. The proposed design is not driven by the design of the online software infrastructure of a concrete subsystem in order to improve the evolution potential of the low level infrastructure. Moreover, the design is logically and technologically decoupled with any controller.

A distributed development will be necessary. Each subsystem will be responsible to customize its own TS leaf. The necessary documentation will be delivered, and a set of configuration management actions have been presented in order to improve the communication of the system evolution and the coordination among trigger subsystem development groups.

The problem that is being identified in this note affects large experiment collaborations in general. Therefore, the proposed system architecture, development methodology and configuration management actions can be used as a solution to be applied in the context of other experiments.

Acknowledgements

The authors would like to acknowledge the many collaborators contributing to the trigger online software infrastructure for their comments and fruitful discussions with them. We are especially grateful for the valuable suggestions of Sergio Cittolin, Johannes Gutleber, Luciano Orsini and Wesley H. Smith. The CMS-internal reviewers Jeremy Mans, Chris Tully and the chairpersons of the CMS Publications Committee and Trigger/DAQ Editorial Board, Roberto Tenchini and Samim Erhan, are thanked for their help in bringing this document into its final form. And last, but not least, we wish to acknowledge the imaginative and open minded ideas of Marc Magrans de Abril.

Acronyms

ACM	Access Control Module
API	Application Program Interface
BC	Bunch crossing
CCI	Control Cell Interface
CERN	Conseil Européen pour la Recherche Nucléaire
CMS	Compact Muon Solenoid
CSC	Cathode Strip Chamber
CVS	Concurrent Versions System
DAQ	Data Acquisition
DB	Data Base
DBWG	CMS Data Base Working Group
DCS	Detector Control System
DT	Drift Tube
ECAL	Electromagnetic Calorimeter
ERM	Error Manager
FDL	Final Decision Logic
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
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
I2O	Intelligent Input/Output
JC	Job Control
LHC	Large Hadron Collider
L1A	Level-1 Accept signal
L1T	Level-1 Trigger
ORCA	Object Oriented Reconstruction for CMS Analysis
OSWI	Online SoftWare Infrastructure
PSI	PVSS SOAP Interface
PVSS	Prozessvisualisierungs- und Steuerungssystem
RC	Run Control
RCMS	Run Control and Monitoring System
RPC	Resistive Plate Chamber
R/O	Readout
SOAP	Simple Object Access Protocol
SRM	Shared Resources Manager
TCS	Trigger Control System
TPG	Trigger Primitive Generator (ECAL, HCAL, HF, DT and CSC)
TS	Trigger Supervisor
TSM	Task Scheduler Module
TTC	Timing, Trigger and Control System
TTS	Trigger Throttle System
URL	Uniform Resource Locator
WSDL	Web Service Description Language
XDAQ	Cross-Platform DAQ Framework
XML	Extensible Markup Language
XSD	XML Schema Document

References

[1] The CMS Collaboration, CMS Technical Proposal, CERN LHCC 94-38 (1994).

[2] The CMS Collaboration, The Trigger and Data Acquisition Project, Vol. I: The Level-1 Trigger, CERN LHCC 2000-038 (2000).

[3] The CMS Collaboration, The Trigger and Data Acquisition Project, Vol. II: Data Acquisition and High-Level Trigger, CERN LHCC 2002-26 (2002).

[4] V. Brigljevic et al., “Run Control and Monitor System for the CMS Experiment”, Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.

[5] The CMS Collaboration, “The Compact Muon Solenoid, Letter of Intent”, CERN/LHCC 1992-3 (1992).

[6] The CDF Collaboration, Proposal for an Upgraded CDF Detector, CDF/DOC/PUBLIC/1172 (1990).

[7] D0 Upgrade proposal (10/18/90); D0 note 1148 (6/18/91); D0 note 1322 (1/13/92); D0 note 1426 (5/20/92).

[8] CMS Trigger / DAQ Group, “CMS L1 Trigger Control System”, CERN CMS Note 2002/033 (2002).

[9] C.-E. Wulz, “Concept of the First Level Global Trigger for the CMS experiment at LHC”, Nucl. Instr. and Meth. **A473** (2001) 231 – 242.

[10] A. Taurok, H. Bergauer, M. Padrta, “Implementation of the First Level Global Trigger for the CMS experiment at LHC”, Nucl. Instr. and Meth. **A473** (2001) 243 – 259.

[11] Web Services Activity, `http://www.w3.org/2002/ws/`

[12] Simple Object Access Protocol (SOAP) 1.1, `http://www.w3.org/TR/soap`

[13] Extensible Markup Language (XML), `http://www.w3.org/XML`

[14] XML Schema, `http://www.w3.org/XML/Schema`

[15] Web Services Description Language (WSDL) 1.1, `http://www.w3.org/TR/wsdl`

[16] I2O Special Interest Group, Intelligent I/O (I2O) Architecture Specification v2.0 (1999).

[17] J. Gutleber, L. Orsini, “Software Architecture for Processing Clusters Based on I2O”, Cluster Computing, Vol. **5**, Issue 1 (2002) 55 – 64.

[18] 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.