Conceptual Design of the CMS Trigger Supervisor

Ildefons Magrans de Abril, *Member, IEEE*, Claudia-Elisabeth Wulz, and João Varela

***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. The functional and nonfunctional requirements, the design, the operational details, and the components needed in order to facilitate a smooth integration of the trigger software in the context of CMS are described.**

***Index Terms*—CMS, online software, trigger supervisor.**

I. 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 or heavy-ion beams with high energies, which are made to collide in CMS and the other experiments located along the accelerator ring, producing a large number of particles. The trigger and the data acquisition are vital components of the CMS 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 sequential 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], which has access to the complete event data. Apart from recording events, the data acquisition system (DAQ) [3] also provides the Run Control and Monitoring System (RCMS) framework [4] for the experiment.

The L1T has to make the decision to accept or reject a collision event for each crossing of the particle bunches which circulate in the LHC. The bunch crossing interval for protons is 25 ns. Depending on luminosity, several interactions of proton constituents occur at each crossing. At the nominal LHC design luminosity of 10^{34} cm^{-2}s^{-1} the interaction rate is therefore close to 1 GHz. The maximum output bandwidth allocated to the L1T is 100 kHz, in accordance with the input capacity limit of the HLT processor farm. The highest rate at which events can be archived by the online computer farm is of the order of 100 Hz.

The L1T has local, regional and global components. Together with the HLT they make up the trigger subsystems. At the bottom end, the Local Triggers for calorimetry and muons, also called Trigger Primitive Generators (TPG), are based on energy deposits in calorimeter trigger towers and track segments in muon chambers, respectively. Regional Triggers combine their information and determine trigger objects such as electron or muon candidates in limited spatial regions, for example in sectors of CMS. Subsequently, the Global Calorimeter and Global Muon Triggers determine the four highest-rank calorimeter and muon objects across the entire experiment. The Global Trigger, the top entity of the level-1 hierarchy, makes the decision to reject an event or to accept it for further evaluation by the HLT. This decision is based on algorithm calculations performed by the Global Trigger logic (GTL) [5], [6] on one hand, and on information from the Trigger Control System (TCS) [7], which takes into account operating conditions and feedback from the CMS subsystems, on the other. Depending on physics search priorities, Global Trigger algorithms have to be selected. The set of algorithms running concurrently at a given time is called trigger menu. Logically the TCS belongs to the Global Trigger, and its central electronics board is physically located in the Global Trigger crate. The TCS allows different subsystems to be operated independently if required. For this purpose the experiment is subdivided into 32 partitions. A partition represents a major component of a subsystem. Each partition must be assigned to a partition group, also called a TCS partition. Within such a TCS partition all connected partitions operate concurrently. For commissioning and testing up to eight TCS partitions are available, which each receive their own level-1 accept decision (L1A) signals. During normal physics data taking there is only one single TCS partition.

In contrast to the Global Trigger including the TCS, the Regional and the Local Triggers, the Trigger Supervisor (TS) is an online software system. Its purpose 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.

Manuscript received July 20, 2005; revised November 25, 2005.

I. Magrans de Abril is with the Institute for High Energy Physics of the Austrian Academy of Sciences, Vienna, Austria, and also with the Electronic Engineering Department, Universidad Autónoma de Barcelona, Barcelona, Spain (e-mail: ildefons.magrans@cern.ch).

C.-E. Wulz is with the Institute for High Energy Physics of the Austrian Academy of Sciences, Vienna, Austria, and also with the University of Technology, Vienna, Austria (e-mail: claudia.wulz@cern.ch).

J. Varela is with LIP—Laboratório de Instrumentação e Física Exp. de Partículas, Lisboa, Portugal, and also with CERN, Geneva, Switzerland, and IST—Instituto Superior Técnico, Lisboa, Portugal (e-mail: joao.varela@cern.ch).

Digital Object Identifier 10.1109/TNS.2006.872631

An intermediate node, lying between the client and the trigger subsystems, which offers a simplified API to perform control, monitoring and testing operations, will ease the design of this client. This layer provides a uniform interface to perform hardware configurations, monitor the hardware behavior or to run tests in which 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 within the broader context of the experiment operation. In this context, the RCMS online framework will be in charge of offering a control window from which 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. Once the TS is accessed through RCMS, a scientist working on a data taking run will be presented with a graphical user interface offering choices to configure, test, run and monitor the trigger system. Configuring includes setting up the programmable logic and physics parameters such as energy or momentum thresholds in the trigger hardware. Predefined and validated configuration files are stored in a data base (DB) and are proposed as defaults. Tests of the trigger system after configuration are optional. Once the TS has determined that the system is configured and operational, a run may be started through RCMS and the option to monitor can be selected. For commissioning periods more options are available in the TS, namely the setting up of different TCS partitions and separate operations of subsystems. It is remarked that all components of the trigger may be operated without the TS and RCMS in case of their temporary unavailability. In such a case, however, compatibility and consistency of operations have to be assured manually to a large extent.

The design of the 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 a period of many years. During this time the hardware and software environments evolve.
- 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 described solution for the trigger system of CMS can be an example for other experiments.

II. Requirements

A. 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 tests 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 of adopting new functionalities required by specific subsystems.

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

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

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. Interconnection and self test operations involve not only the trigger subsystems but also the subdetectors themselves (Section III-B3).

4) Monitoring: The TS interface must enable the monitoring of the necessary information that assures the correct functionality of the trigger subsystems (e.g., measurements of trigger rates and efficiencies, simulations of the Level-1 Trigger hardware running in the HLT), subsystem specific monitoring data (e.g., data read through spy memories), and information for synchronization purposes.

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 run control sessions, running concurrently, will need to access the trigger system. Therefore, it is necessary that the TS coordinates the access to the common resources (e.g., the TCS, the trigger subsystems and the configuration data base). In addition, it is necessary to control the access to the trigger system hierarchically in order to determine which users/entities (controllers) can have access to it and what privileges they have. A complete access control protocol has to be defined that will include identification, authentication, and authorization processes. Identification includes the processes and procedures employed to establish a unique user/entity identity within a system. 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 is the process of deciding if a requesting user/entity is allowed to have access to a system service. A hierarchical list of users with the corresponding level of access rights as well as the necessary information to authenticate them will be maintained in the configuration data base. The lowest-level user is only allowed to monitor. A medium-level user, such as a scientist responsible for the data taking during a running period of the experiment, may manage partition setups, select predefined trigger menus and change thresholds, which are written directly into registers on the electronics boards. In addition to all the previously cited privileges the highest-level user or super user is allowed to reprogram logic and change internal settings of the boards. In addition to coordinating the access of different users to common resources, the TS must also ensure that operations launched by different users are compatible.

6) Hierarchical Start Up Mechanism: In order to maximize subsystem independence and client decoupling (Section II-B3), a hierarchical start up mechanism must be available (Section III-B5 describes the operational details). As will be described later, the TS is organized in a tree-like structure, with a central node and several leaves. The first run control session or controller will be responsible for starting up the TS central node, and in turn this will offer an API that provides start up of the TS leaves and the online software infrastructure of the corresponding trigger subsystem.

7) Logging Support: The TS must provide logging mechanisms in order to support the users carrying out troubleshooting activities in the event of problems. Logbook entries must 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 offline use.

8) Error Handling: An error management scheme, compatible with the global error management architecture, will be necessary. It must provide a standard error format, and remote error handling and notification mechanisms.

9) User Support: A graphical user interface (GUI) will be provided. This will allow a standalone operation of the TS. It will also help the user to interact with the TS framework and to visualize the state of a given operation or the monitoring information. 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.

An adequate online help facility shall be available 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.

B. Nonfunctional Requirements

1) Low-Level Infrastructure Independence: The design of the TS should be independent of the online software infrastructure (OSWI) of any subsystem as far as possible. 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) Subsystem Control: The TS should offer the possibility of operating a concrete trigger subsystem. Therefore, the design should be able to 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.

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 a 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 evolution potential and decrease the maintenance effort 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.

4) Multi User: During the commissioning and maintenance phases, several run control sessions will be running concurrently. Each of them will be responsible for operating a different TCS partition. In addition, the TS should allow standalone operations (not involving the RCMS 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.

5) Remote Operation: The CMS experiment is developed and will be operated by more than one hundred institutions located all over the world. Unlike in the past, most scientists can in general not be present in person at the experiment location during data taking and also during commissioning, but have to operate and supervise their systems remotely. The possibility to program and operate the trigger components remotely is especially important because the trigger directly affects the quality of the recorded data.

6) Interface Requirements: In order to facilitate the integration, the implementation and the description of the controller-TS interface a web service based approach [8] should be followed. The chosen communication protocol to send commands and state notifications should be the same as for most CMS subsystems, and especially the same as already chosen for run control, data acquisition and slow control. Therefore Simple Object Access Protocol (SOAP) [9] and the representation format Extensible Markup Language (XML) [10] for exchanged data should be selected. The format of the transmitted data and the SOAP messages is specified using the XML schema language [11], and the Web Services Description Language (WSDL) [12] is used to specify the location of the services and the methods the service exposes. To overcome the drawback that XML uses a textual data representation, which causes much network traffic to transfer data, a binary serialization package provided within the CMS online software project and I2O messaging [13] could be used for devices generating large amounts of real-time data.

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

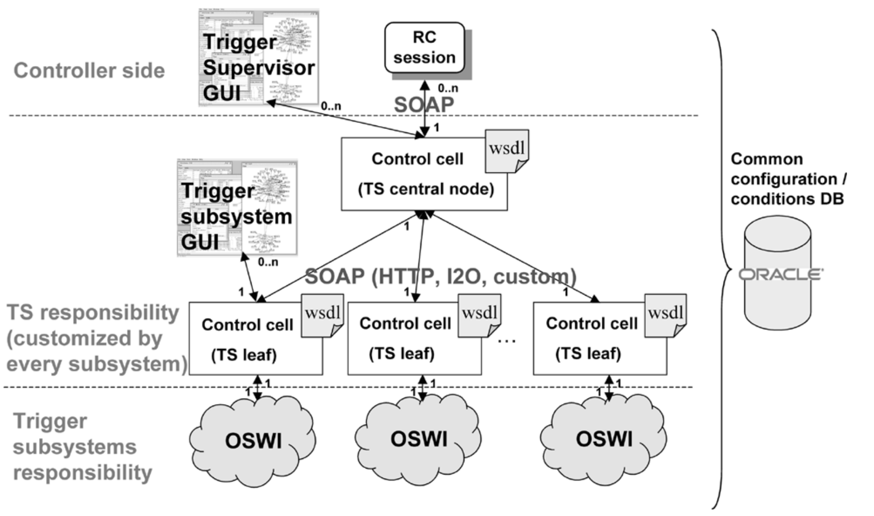

Fig. 1. Architecture of the Trigger Supervisor. A central node and customizable leaves for each subsystem are the core elements.

interface will not block until the completion of the requested operation. The available operations in each node of the TS tree can be seen as a set of different finite state machines. A GUI must be provided with the TS, which shows the operator, for each started operation, the current state, the available actions that can be executed from that state, and a time bar that displays the expected time for completion of a given action.

The OSWI of the trigger subsystems will likely be installed inside a private network. Therefore, it will be necessary to provide access to such networks.

III. Design

The TS architecture is composed of a central node in charge of coordinating the access to the different subsystems, namely the trigger subsystems and subsystems concerned with the interconnection test service (Section III-B3), and a customizable TS leaf (Section IV-B) 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.

Each node of the TS can be accessed independently, fulfilling the requirement outlined in Section II-B2. The available interfaces and location for each of those nodes are defined in a WSDL document. Both the central node and the TS leaves are based on a single common building block, the *control cell*. Each subsystem group will be responsible for customizing a control cell and keeping 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 II-B1). 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 II-B3). In addition, the distributed nature of the TS design will facilitate a clear separation of responsibilities and a distributed development. 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 cross-platform data acquisition framework [14], [15] 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 homogeneous.
- 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.

A. *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. 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 II-B2), 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 (e.g., an I2O interface will be implemented). Each control

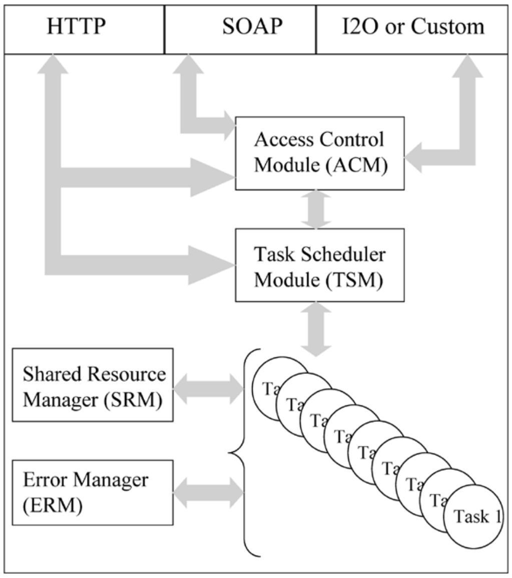
HTTP
SOAP
I2O or Custom
Access Control Module (ACM)
Task Scheduler Module (TSM)
Shared Resource Manager (SRM)
Error Manager (ERM)
Task 1

Fig. 2. Architecture of the control cell, the common building block of all TS nodes.

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 for identifying and authenticating every user or entity (controller) attempting to access, and for providing 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 of managing the command requests and forwarding 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 for preventing the launching of 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 operations could change the available interface of the control cell. In that case, the corresponding WSDL should be updated.
- Shared Resources Manager (SRM): This module is in charge of coordinating access to shared resources (e.g., the configuration data base, 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 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. Both the error format and the remote error notification mechanism will be based on the global CMS distributed error handling scheme.

The control over what operations can be executed is distributed among the ACM for user access level control (e.g., a user with monitoring privileges cannot launch a self test operation), the TSM for 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 is allowed to set up the requested configuration). More details are given in Section III-B1.

B. Trigger Supervisor Services

The Trigger Supervisor services are the final functionalities offered by the TS. 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 leaves. 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 leaves, and how the services emerge from the collaboration of these distributed operations. It is remarked that a control cell operation is always a FSM.

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 programmable logic devices of the electronics boards. The configuration service requires the collaboration of the central node of the TS and all the TS leaves. 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 while the FSM is in a transition state. If applicable, an error state is invoked from the transition state. Fig. 4 shows how the different nodes of the Trigger Supervisor collaborate in order to fully configure the trigger system.

A key—a name that uniquely identifies the configuration of a given system—is assigned to each node. Each key maps into a primary key in the data base table that contains the configuration information of the system. The sequence of steps that a controller of the TS should follow in order to properly use the configuration service is as follows.

- 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_keys,” one per trigger subsystem that is to be configured. The “`Configure(TS_key)`” command initiates the configuration operation in the relevant TS leaves. The configure command in the configuration operation of each TS leaf will check whether or not the user is allowed to set the

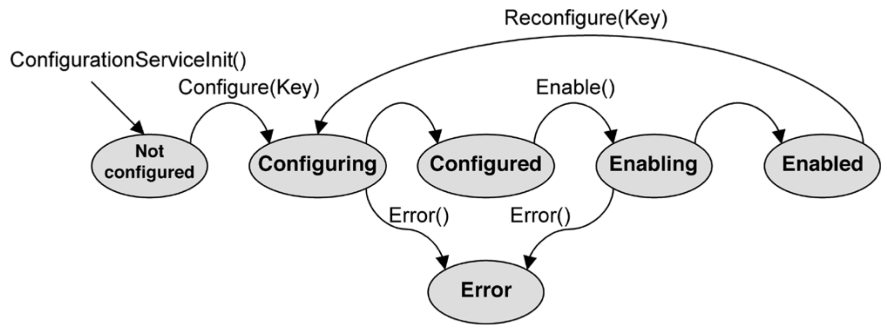

Fig. 3. Configuration operation. Such an operation has to be implemented by each control cell.

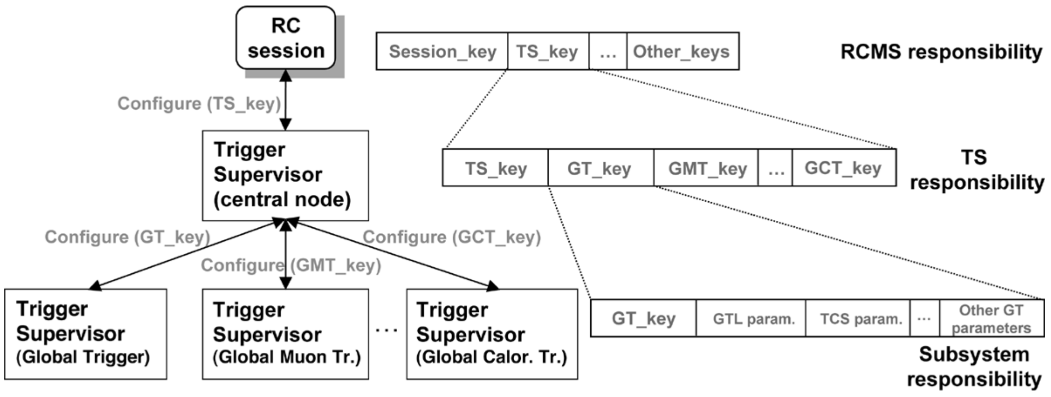

Fig. 4. Configuration service. A configuration is identified by a unique key.

configuration identified by a given "trigger_key." This means that each trigger subsystem has the full control over who and what can be configured. This also means that the list of users in the central node of the TS will be replicated in the TS leaves.

- Once the configuration operation of the TS leaves reaches the "Configured" state, the configuration operation in the central node of the TS jumps to the "Configured" state.
- Send an "`Enable()`" command. This fourth step is just a switch-on operation.

From the point of view of the trigger system, everything is ready to run the experiment once the configuration operation reaches the "Enabled" state.

Each trigger subsystem 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_keys," 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 III-D, the configuration data base is common to both the central node of the TS and the TS leaves. Each trigger subsystem will be responsible for populating the configuration data base and to assign key identifiers to sets of configuration parameters.

2) Reconfiguration: This section complements Section III-B1. A reconfiguration of the trigger system may become necessary, for example if thresholds have to be adapted due to a change in luminosity conditions. The new configuration table must be propagated to the filter farm, as it was required in Section II-A2. The following steps show how a controller of the TS should behave in order to properly reconfigure the trigger system using the configuration service.

- Once the trigger system is configured, the configuration operation in the central node of the TS will be in the "Enabled" state.
- Send a "`Reconfigure(key)`" command. The following steps show how this command behaves.
  —Stop the generation of L1A signals.
  —Send a "`Configure(TS_key)`" command as in Section III-B1, and
  —Jump to the state "Configured."
- The controller is also responsible for propagating the configuration changes to the filter farm hosts in charge of the HLT and the trigger simulation through the configuration/conditions data base.
- 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 trigger simulation. This command will be in charge of resuming the generation of L1A signals.

Run control is in charge of coordinating the configuration of the TS and the HLT. There is no special interface between the central node of the TS and the HLT.

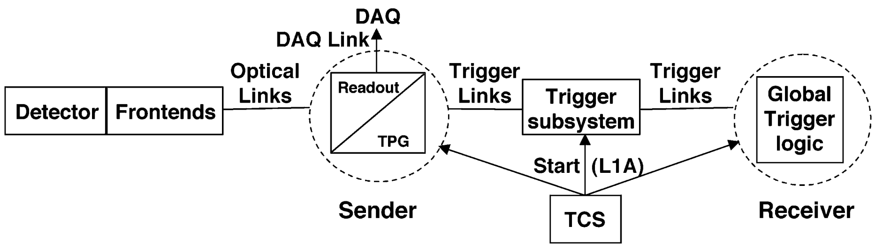

Fig. 5. Typical scenario of an interconnection test. The interconnections of the Trigger Primitive Generators and the Global Trigger logic are tested.

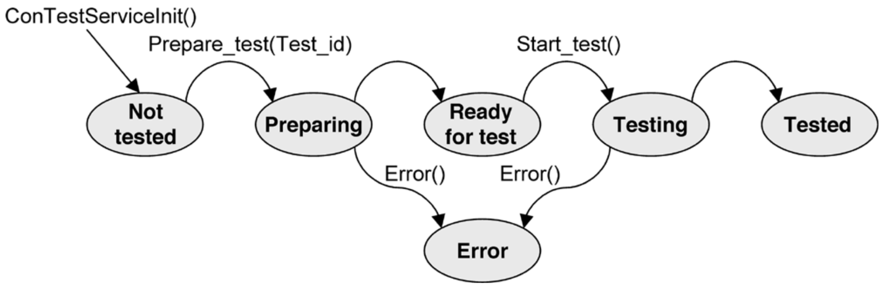

Fig. 6. Interconnection test operation. Each control cell must implement this type of operation.

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

The self test 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 leaves. Each control cell involved will implement the same self test operation. The self test operation running in each control cell is a FSM with only 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.

- Send a "`SelfTestServiceInit()`" command. Once the self test operation is initiated, the operation reaches the "halt" state (initial state).
- 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 used to follow the above sequence for each TS leaf, and collect all error reports coming from the TS leaves. In the case of a 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).

The interconnection test 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 for participants involved in an interconnection test. The example shows the interconnection test of the Trigger Primitive Generators and the Global Trigger logic.

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

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

- Send a "`ConTestServiceInit()`" command.
- 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:
  - —Retrieve from the configuration data base the relevant information for the central node of the TS.
  - —Send a "`ConTestServiceInit()`" command to sender/s and receiver/s.
  - —Send "`Prepare_test()`" command to sender/s and receiver/s.
  - —Wait for "Ready_for_test" signal from all senders/receivers.
- Once the operation reaches the "Ready" state, the next step is to send a "`Start_test()`" command.
- Wait for results.

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

- Retrieve from the configuration data base the relevant information for the leaf (e.g., which role: sender or receiver, test vectors to be sent or to be expected).
- Send a "Ready_for_test" signal to the central node of the TS.
- Wait for the "`Start_test()`" command.

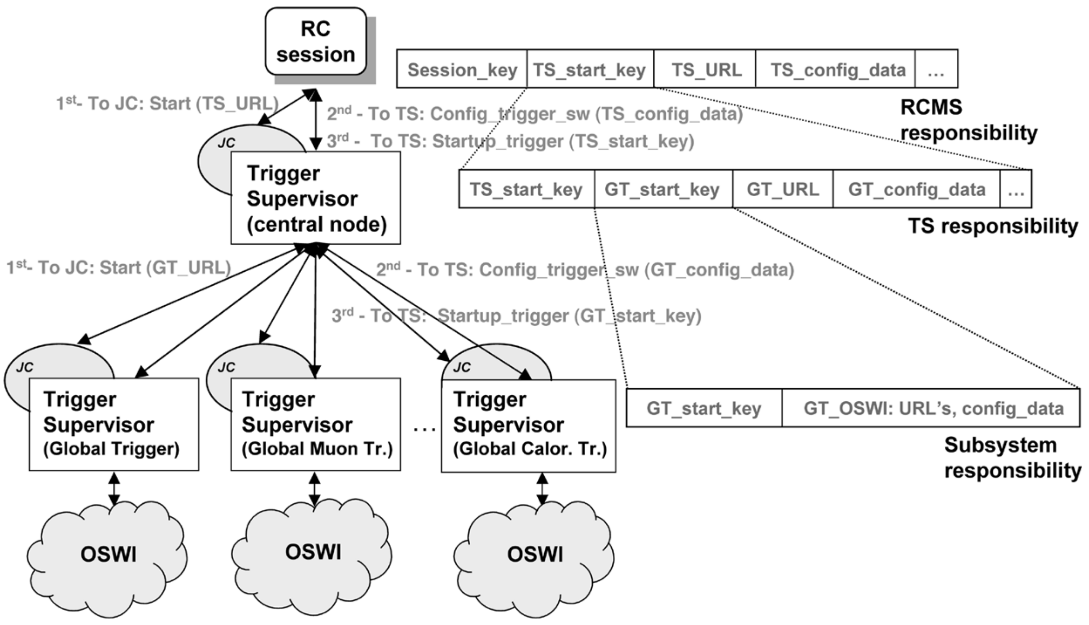
RC session
Session_key | TS_start_key | TS_URL | TS_config_data | ...
RCMS responsibility
1st- To JC: Start (TS_URL)
2nd - To TS: Config_trigger_sw (TS_config_data)
3rd - To TS: Startup_trigger (TS_start_key)
JC
Trigger Supervisor (central node)
TS_start_key | GT_start_key | GT_URL | GT_config_data | ...
TS responsibility
1st- To JC: Start (GT_URL)
2nd - To TS: Config_trigger_sw (GT_config_data)
3rd - To TS: Startup_trigger (GT_start_key)
Trigger Supervisor (Global Trigger)
Trigger Supervisor (Global Muon Tr.)
Trigger Supervisor (Global Calor. Tr.)
GT_start_key | GT_OSWI: URL's, config_data
Subsystem responsibility
OSWI
OSWI
OSWI

Fig. 7. Start up service. Three steps identified by keys have to be executed.

- 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).

In contrast to the configuration service, the central node of the TS can already check whether a given user can launch interconnection test operations. However, the TSM of each TS leaf will still be in charge of checking whether acting as a sender/receiver is in conflict with an already running operation. Each subdetector must also customize a control cell in order to facilitate the execution of interconnection tests that involve the TPG modules.

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 leaves.

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 facilitates the implementation of an alarm mechanism, part of which can be incorporated in the general alarm system provided through the RCMS framework.

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

5) Start Up: From the point of view of a controller (run control session or standalone client), the whole trigger system is one single resource, which can be started by 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 these steps.

- Send a "`Start(TS_URL)`" command to the job control (JC) daemon in charge of starting up the central node of the TS, where 'TS_URL' identifies the Uniform Resource Locator from where the compiled central node of the TS can be retrieved.
- 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.
- Send a "`Startup_trigger(TS_start_key)`" command to the central node of the TS. This command will send the same sequence of three commands to each TS leaf, but now the command parameters are retrieved from the configuration data base register identified with the "TS_start_key" index. The "`Startup_trigger(TSLeaf_start_key)`" command that is received by the TS leaf is in charge of starting 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 leaves) 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.

C. 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 II-A9. Two different interfaces will be provided.

- HTML: The control cell will provide an HTTP interface that will allow full operation of the control cell and visualization of the state of any running operation. The HTTP interface will provide an additional entry point to the control cell (Section III-A), 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.

D. Configuration/Conditions Data Base

A common configuration/conditions data base approach, as it is shown in Fig. 1, will be used by all the trigger subsystems and the TS. Different sets of firmware for the trigger electronics boards and default parameters such as thresholds will be predefined and stored in the data base. The information will be validated with respect to the actual hardware limitations and compatibility between different components.

The general CMS data base infrastructure, which the TS will use, includes 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 the specific data base structures for storing configuration data, access control information and interconnection test parameters. Custom GUIs to populate these structures should also be delivered.

IV. Deliverables and Configuration Management

The development of the TS will require the distribution of its implementation among the trigger subsystems and the TS. The skeleton of the basic control cell, presented in Section III-A, will be delivered to every trigger subsystem. Each subsystem will be responsible for its customization. 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. Configuration management is the discipline applying technical and administrative controls to identification and documentation of physical and functional characteristics of configuration items, changes to characteristics of those configuration items, recording and reporting of change processing and implementation of the system. The term "software configuration management" will be used in the TS context in the following, in order to distinguish it from the detector and trigger hardware detector configuration management.

A. 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 III-A).
- Pluggable set of default operations: FSMs for configuration, monitoring and testing and proposed API of commands to be filled by subsystems (Section III-B).
- Skeleton of a customizable GUI (Section III-C)

B. Development Methodology

A template TS leaf, which has to be customized by each subsystem, is provided. Fig. 8 shows three 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 access to the VME crate.

The scenarios for the customization of the TS leaf are as follows.

- Web service access to the OSWI: in this case the customization of the control cell will use the web service API to access the hardware.
- XDAQ-based OSWI: in this case one can either use the SOAP API to access the XDAQ services or bind the control cell and the OSWI to the same XDAQ executive. In the second case, it is possible to use the XDAQ infospace for the communication between 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 has 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.

C. 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.

- It facilitates the exact replication of the trigger system.
- It improves the communication of the software evolution.
- It facilitates the coordination among trigger subsystem groups.
- It eases the resource sharing among trigger subsystem groups.

An exhaustive software configuration management would require tracking several parameters. On the other hand, a more

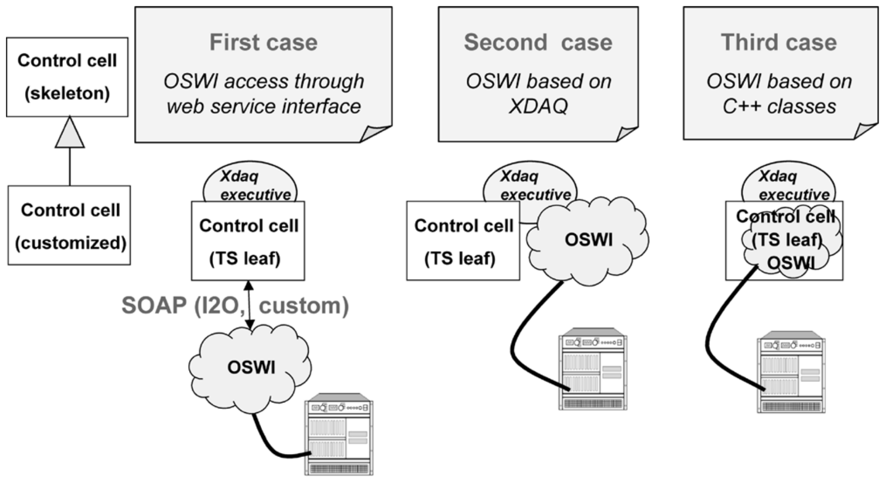

Fig. 8. Integration scenarios. Three scenarios for the customization of a TS leaf are possible.

suitable and realistic approach is to adopt different actions in a progressive way.

A common CVS (Concurrent Versions System) repository for all the online software infrastructure of the trigger has been created, which facilitates the production and coordination of trigger software releases. A generic Makefile has been adopted to homogenize the build process of the trigger software. This allows a more automatic deployment of the trigger online software infrastructure, and prepares it for integration with the online software framework of the DAQ.

V. Summary and Conclusion

The requirements for the Trigger Supervisor have been identified and a suitable architecture has been developed. It is composed of a central node in charge of coordinating the access to the different subsystems, and a customizable TS leaf for each of them, which offers a well defined interface to the central node. The concept 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 from any controller.

A distributed development will be necessary. Each subsystem will be responsible for customizing its own TS leaf. 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. A working prototype of the TS containing the principal components but operating with only one subsystem has been provided. It will be further developed for integration tests of the trigger electronics and a first data taking run of the CMS experiment with cosmic rays in the year 2006.

The problem of a distributed online software system affects large experiment collaborations in general. Therefore, the described system architecture, development methodology and configuration management actions can be used as a solution that is applicable to other experiments.

Acknowledgment

The authors would like to thank the many collaborators contributing to the trigger online software infrastructure for their comments and fruitful discussions with them; S. Cittolin, J. Gutleber, L. Orsini, and W. H. Smith for valuable suggestions; the CMS-internal reviewers J. Mans, C. Tully, and the chairpersons of the CMS Publications Committee and Trigger/DAQ Editorial Board, R. Tenchini and S. Erhan, for their help in bringing the manuscript into its final form. Last, but not least, the authors would like to thank M. Magrans de Abril for imaginative and open-minded ideas.

References

[1] The CMS Collaboration, CMS Tech. Proposal, CERN LHCC 94-38, 1994.
[2] The CMS Collaboration, The Trigger and Data Acquisition Project, The Level-1 Trigger, CERN LHCC 2000-038, vol. I, 2000.
[3] The CMS Collaboration, The Trigger and Data Acquisition Project, Data Acquisition and High-Level Trigger, CERN LHCC 2002-26, vol. II, 2002.
[4] V. Brigljevic *et al.*, "Run control and monitor system for the CMS experiment," presented at the Computing in High-Energy and Nuclear Physics Conf., La Jolla, CA, Mar. 24–28, 2003.
[5] C.-E. Wulz, "Concept of the first level global trigger for the CMS experiment at LHC," *Nucl. Instrum. Meth. A*, vol. 473, pp. 231–242, 2001.
[6] A. Taurok, H. Bergauer, and M. Padrta, "Implementation of the first level global trigger for the CMS experiment at LHC," *Nucl. Instrum. Meth. A*, vol. 473, pp. 243–259, 2001.
[7] CMS L1 Trigger Control System, CMS Trigger/DAQ Group, CERN CMS Note 2002/033, 2002.
[8] Web Services Activity [Online]. Available: http://www.w3.org/2002/ws/
[9] Simple Object Access Protocol (SOAP) 1.1 [Online]. Available: http://www.w3.org/TR/soap
[10] Extensible Markup Language (XML) [Online]. Available: http://www.w3.org/XML
[11] XML Schema [Online]. Available: http://www.w3.org/XML/Schema
[12] Web Services Description Language (WSDL) 1.1 [Online]. Available: http://www.w3.org/TR/wsdl
[13] I2O Special Interest Group, Intelligent I/O (I2O) Architecture Specification v2.0, 1999.
[14] J. Gutleber and L. Orsini, "Software architecture for processing clusters based on I2O," *Cluster Comput.*, vol. 5, no. 1, pp. 55–64, 2002.
[15] J. Gutleber, S. Murray, and L. Orsini, "Toward a homogeneous architecture for high-energy physics data acquisition systems," *Comput. Phys. Commun.*, vol. 153, no. 2, pp. 155–163, 2003.