INTERCONNECTION TEST FRAMEWORK FOR THE CMS LEVEL-1 TRIGGER SYSTEM

J. Hammer, CERN, Geneva, Switzerland
M. Magrans de Abril, Wisconsin University, Madison, Wisconsin, U.S.A.
C-E. Wulz, Austrian Academy of Sciences, Vienna, Austria

Abstract

The Level-1 Trigger Control and Monitoring System is a software package designed to configure, monitor and test the Level-1 Trigger System of the Compact Muon Solenoid (CMS) experiment at CERN's Large Hadron Collider. It is a large and distributed system that runs over 50 PCs and controls about 200 hardware units.

The objective of this paper is to describe and evaluate the architecture of a distributed testing framework – the Interconnection Test Framework (ITF). This generic and highly flexible framework for creating and executing hardware tests within the Level-1 Trigger environment is meant to automate testing of the 13 major subsystems interconnected with more than 1000 links. Features include a web interface to create and execute tests, modeling using finite state machines, dependency management, automatic configuration, and loops. Furthermore, the ITF will replace the existing heterogeneous testing procedures and help reducing both maintenance and complexity of operation tasks.

INTRODUCTION

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

Since millions of collisions of protons occur every second and only a small fraction of these can provide insight into new physics, events have to be selected on-line according to their properties. This is done by the trigger system, a vital part of the experiment.

The trigger system is organized in two levels: (1) The Level-1 Trigger (L1T) [2] is a custom-designed, largely programmable electronic system which preselects the most interesting collisions for further evaluation. These collisions are then examined and – if selected – stored permanently by (2) the High-Level Trigger (HLT) [3], which consists of a farm of industrial processors.

The Trigger Supervisor Framework

The L1T is configured, tested and monitored using the Level-1 Trigger Control and Monitoring System – a large and distributed system that runs over 50 PCs and controls about 200 hardware units (~ 6000 boards). Each of the components of this system is based on the Trigger Supervisor (TS) Framework [4], written in C++, providing a web interface utilizing AJAX.

The TS architecture is composed of a hierarchical tree of nodes, where the central (i.e. the top) node is in charge of coordinating the access to the subsystems [5]. Each node is accessible by a well-defined interface based on the Simple Object Access Protocol (SOAP) [6], and can run one or more commands and operations simultaneously. Operations are stateful objects that use transitions to move between different states of their internal finite state machine (FSM) – which can be defined freely.

The communication hierarchy is strictly top-down – following the request-response model – with subsystems not even knowing their supervisor nodes. Fig. 1 shows an example system setup.

Need for an Interconnection Test Framework

With more than 1000 links connecting the about 200 hardware units, thorough testing is crucial. So far, the various subsystems have used custom scripts for testing – controlling the TS nodes via SOAP commands. Needless to say, this leads to a lot of duplicated effort with a high level of maintenance required. Even more important, this approach does not allow the centrally controlled execution of tests (in particular by non-experts!), nor to execute them automatically at specific events.

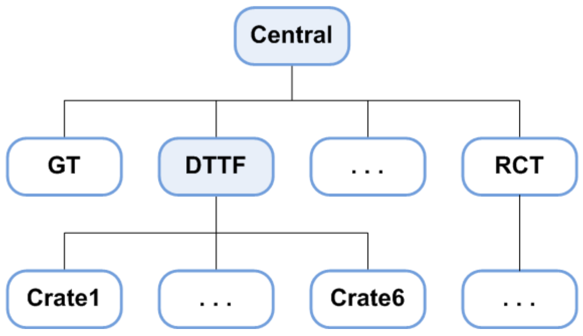


Figure 1: An example Level-1 Trigger system setup.

Instead, local test methodologies heavily rely on the respective subsystem experts and time-consuming coordination between them and the global run control operators. All these factors lead to unnecessary system downtime and a waste of precious manpower, both undesirable consequences for such a huge experiment involving large resources of manpower and investment.

A much better solution would be to provide an interconnection test framework on top of the existing TS framework. This could make use of the already available infrastructure and facilities, and provide additional common functionality for distributed testing. However, although the basic idea was already stated in [6], a sophisticated implementation was yet to be developed.

Requirements and Constraints

Among the requirements:

- Main focus on simplicity: The users should need as little effort as necessary to use the ITF (simple API, little setup/configuration, …).
- Tests can be run both locally and centrally controlled.
- Tests can be run in different network environments (with or without database; with various and even fake top nodes, …).
- Unlimited subsystem hierarchy depth.
- Shall be very flexible to allow a large range of diverse tests (including self-tests).

Imposed constraints were:

- Shall be embedded into the existing Trigger Supervisor environment, i.e. the tests shall be implemented using TS operations.
- Shall allow both interactive and scripted execution.

ARCHITECTURE

Overview

The Interconnection Test Framework is built on top of the Trigger Supervisor Framework, and therefore fully embedded into the existing environment. As a result, the ITF is highly TS specific, using the available API to communicate with subsystems and drive operations. Figure 2 shows the basic architecture:

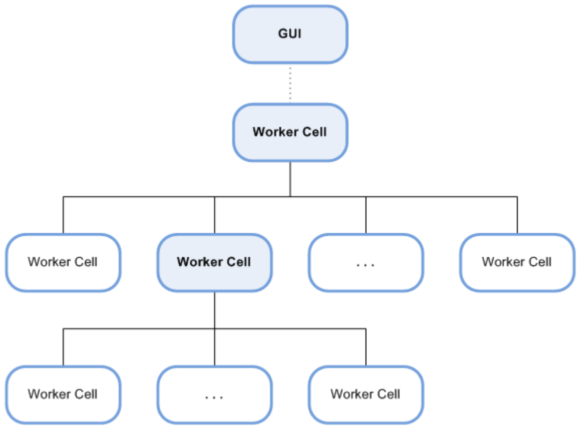


Figure 2: The architecture of the Interconnection Test Framework.

Each node runs one Interconnection Test operation – a sub class of the framework's base class, with the concrete functionality implemented by the user. Any node will automatically act as a supervisor once there are sub nodes defined for it.

Which and how many nodes (i.e. systems) are involved depends on the test case – essentially an XML document – that defines all the necessary details for a specific test (e.g. user-defined parameters).

Service Discovery (ITF-SD)

Much effort was spent on simplicity. Wherever possible, the "convention over configuration" paradigm was applied to reduce both the required initial effort to create a test case as well as the necessary maintenance.

One core feature that facilitates this strategy is the Service Discovery (ITF-SD). It allows to automatically detect all available test operations in the network which are reachable from the start node. Based on this information is automatically creates test cases without having to define the sub nodes – exploiting the already defined system hierarchy.

This eliminates the need to maintain the test cases when the topology of the network (or just the name of a subsystem) changes. Both partly pre-defined (auto-detecting a specific sub tree only) and on-the-fly test cases (auto-detecting the entire test tree) are possible.

The Finite State Machine (FSM)

The finite state machine is the main building block of a test operation. It defines the available system states and the legal transitions to move between them, and serves as the common specification for all test operations. Consequently, the FSM should be quite flexible to allow a diverse set of test types – starting from self-tests that would need only one single transition, to complex distributed tests that require loops with plenty of synchronization between the nodes.

Figure 3 shows the finite state machine that is used for the Interconnection Test Framework. It features six transitions: setup, prepare, execute, analyze, next and summarize. Each test has at least one loop, the number being either defined statically in the test case definition or set dynamically at runtime (by any node).

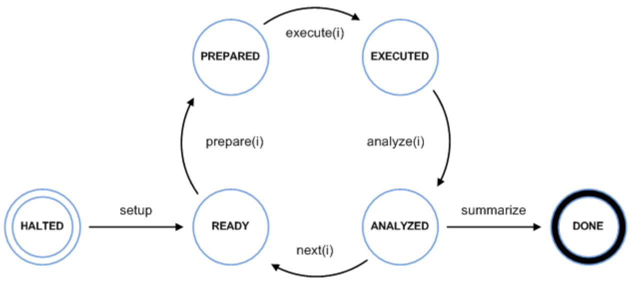


Figure 3: The finite state machine for all test operations.

FSM Pruning

As shown in the previous section, the FSM features a loop. Each loop provides four different transitions – prepare, execute, analyze and next. Each of these transitions has to be called in every subsystem in every loop. However, most test operations will not need that many steps, e.g. execute and analyze only might be sufficient. Nevertheless, the empty transitions have to be called as well – leading to a significant performance penalty when many loops are required (not to mention that every connection to be made has a risk to fail).

To get rid of these disadvantages while still providing a flexible and unique FSM, FSM pruning (or transition-skipping) is applied. During the first loop the framework automatically detects which transitions are empty (i.e. not in use) and uses this information to skip those transitions in the following loops. This works on a per sub-tree basis. The only thing the developer has to do is not to implement the corresponding transition.

Figure 4 shows the actual FSM that will be utilized in case the prepare transition is empty.

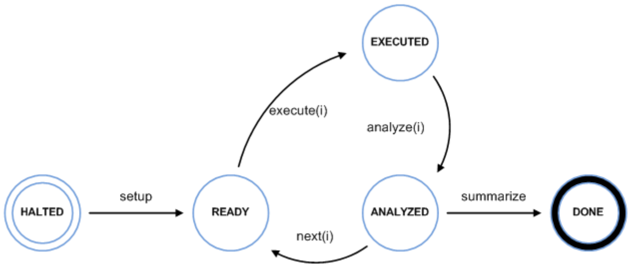


Figure 4: FSM Pruning: Skipping the prepare transition.

Sequences

Sequences are used to define the order in which a transition should be executed in different subsystems of the same level of the hierarchy. Their main purpose is to define dependencies. If nothing is specified explicitly, the default rule is to execute everything in parallel.

Shared Memory

The communication in the Trigger Supervisor system is strictly top-down. However, for some tests it is convenient or even necessary to communicate with other (e.g. the neighbor) nodes.

Therefore, the ITF provides a simple shared memory, based on name/value pairs. A name may also refer to a file on the local hard disk – the framework will automatically take care of syncing it to any node that needs the file.

The Test Result

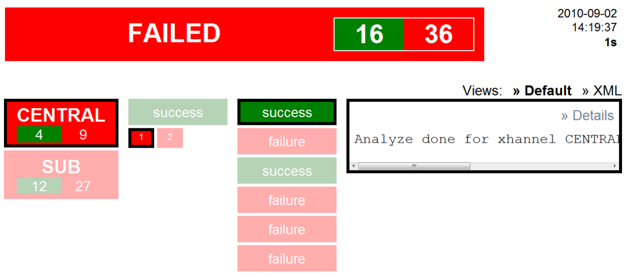
The basic building block of a test result is the ResultItem. A ResultItem – either of type success or failure – may contain a message and additional attributes (which allows structured information). Neutral items (so-called properties) are possible too to include general information, e.g. the current state of the hardware.

It is also possible to store detailed result data in separate files. A test may generate as many ResultItems as required. To issue a failure with an empty message and two attributes all that is required is something like

```
failure("").attributes()
        .add("source", boardID)
        .add("expected", pattern));
```

(Custom) Views / Analyzers

In order to visualize the test result in a user-friendly way, result views are provided. With the default view (Fig. 5) the ResultItems of all systems and loops can easily be examined. By using AJAX even huge test results with thousands of items can be browsed through without putting a heavy load on the browser.



```
Analyze done for xhannel CENTRAI
```

Figure 5: The framework's default result view.

In order to provide higher flexibility and facilitate a wide variety of test types, custom result views / analyzers are possible as well (like the one in Fig. 6). This allows interpreting the result in a specific way and, in particular, visualizing it according to the needs of that type of test.

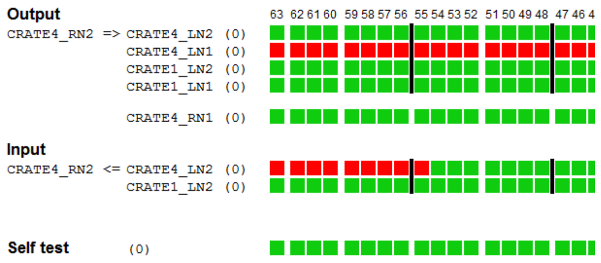


Figure 6: A custom result view.

Central History

All test results are automatically stored into the database to provide a central history. Therefore, the results can be evaluated and compared to previous results at any later time. As one of the goals of the ITF is to be database independent, this will fail silently in case no database is available.

As an additional feature automatic logging of all actions and (sub) results is provided to help debugging issues.

RELATED WORK

The CMS experiment with its many custom-made hardware units is, well, unique. Therefore, the Trigger Supervisor - a system which is responsible for centrally controlling all those units – already had to be developed in-house as there was no other solution available. Thus, the main constraint was that the Interconnection Test Framework must run within the Trigger Supervisor environment.

As a result, there was no other solution than to develop the ITF from scratch for the specific needs of the CMS Level-1 Trigger, after gathering the requirements from different subsystem groups. The basic idea has already been stated in [7], including an original, loop-free version of the finite state machine.

In the wider sense, all the XUnit frameworks can be considered related work. However, although some concepts are similar and ideas were derived in particular from JUnit [8], those are frameworks for unit testing and not for distributed testing.

CONCLUSION

This paper introduced the Interconnection Test Framework for the CMS Level-1 Trigger System, a solution for testing connections between several nodes of a distributed system. The framework is built on top of the Trigger Supervisor framework, and allows executing transitions with arbitrary actions synchronously on several systems.

The excellent flexibility and extensibility of the Interconnection Test Framework was shown – both a wide variety of test types as well as custom result analyzers are possible. Even more important is that the framework tries to reuse many concepts of the underlying Trigger Supervisor framework, enabling the test developers to start right away by using already familiar knowledge.

However, this also means that the concrete implementation of the ITF is applicable for our environment only. Nevertheless, the concepts and the conclusions are hopefully useful for others as well.

Another limitation is due to the fact that XML is quite verbose, which limits the amount of data that can be stored without affecting performance. With the current implementation, about 10000 result items is a reasonable limit (could be improved if necessary).

Still, XML is an ideal choice for both the configuration and the result as it is highly flexible and easy to maintain (in contrast to e.g. JSON). It makes development so much faster if you are able to add attributes or entire trees on-the-fly without having to change anything in your code.

Simplicity is the main prerequisite for user acceptance, and thus the key ingredient for making the framework successful. This was achieved by reusing the concepts of the TS framework and by keeping the required test configuration to a bare minimum with features like the Service Discovery.

The second critical factor is performance. After some analysis we figured out the minimal Finite State Machine that would be needed to implement our tests. As four steps per loop would slow down many tests, we decided to automatically prune the FSM instead of providing several versions. This makes life easier for the developers but still allows high performance.

Although the framework is constantly being developed further, it already provides a stable environment for various kinds of tests in the CMS Level-1 Trigger system where it proved to be immediately applicable for real-world use cases.

ACKNOWLEDGMENT

We would like to thank Christian Hartl and Thomas Themel for providing ideas and valuable feedback. Franz Mittermayr developed the original versions of the test configuration and execution user interfaces.

REFERENCES

[1] CMS Collaboration, “CMS Technical Proposal”, CERN/LHCC 94-38, 1994.
[2] CMS Collaboration, “The TriDAS Project – The Level-1 Trigger Technical Design Report”, CERN/LHCC 2000-38, 2000.
[3] CMS Collaboration, “The TriDAS Project – Data Acquisition and High-Level Trigger Technical Design Report”, CERN/LHCC 2002-26, 2002.
[4] I. Magrans de Abril, C-E. Wulz, J. Varela, “Concept of the CMS Trigger Supervisor”, IEEE Trans. Nucl. Sci. Vol. 53 Nr. 2, 474-483, 2006.
[5] I. Magrans de Abril and M. Magrans de Abril, “The CMS Trigger Supervisor Project”, IEEE Nuclear Science Symposium Conference Record, Puerto Rico, 23-29 October, 2005.
[6] Simple Object Access Protocol, http://www.w3.org/TR/soap.
[7] I. Magrans de Abril, “The CMS Trigger Supervisor: Control and Hardware Monitoring System of the CMS Level-1 Trigger at CERN”, Doctoral Thesis, Universitat Autonoma de Barcelona, 2008.
[8] The JUnit Framework, http://www.junit.org.