A RULE-BASED VERIFICATION AND CONTROL FRAMEWORK IN ATLAS TRIGGER-DAQ

A. Kazarov, A. Corso-Radu, G.Lehmann Miotto, J.E. Sloper[1], CERN, Geneva, Switzerland
Yu. Ryabov, Petersburg NPI, Gatchina, Russian Federation

1 on leave from University of Warwick, Coventry, England

Abstract

In order to meet the requirements of ATLAS data taking, the ATLAS Trigger-DAQ system is composed of O(10000) of applications running on more than 2600 computers in a network. With such system size, s/w and h/w failures are quite frequent. To minimize system downtime, the Trigger-DAQ control system shall include advance verification and diagnostics facilities. The operator should use tests and expertise of the TDAQ and detectors developers in order to diagnose and recover from errors, if possible automatically.

The TDAQ control system is built as a distributed tree of controllers, where behaviour of each controller is defined in a rule-based language allowing easy customization. The control system also includes verification framework which allow users to develop and configure tests for any component in the system with different levels of complexity. It can be used as a stand-alone test facility for a small detector installation, as part of the general TDAQ initialization procedure, and for diagnosing the problems which may occur during the run time.

The system is currently being used in TDAQ commissioning at the ATLAS pit and by subdetectors for stand-alone verification of the hardware before it is finally installed.

The paper describes the architecture and implementation of TDAQ control system with more emphasis on the new features developed for the verification framework, features requested by users during it's exploitation in real environment.

INTRODUCTION

The paper describes components of the Run Control and Verification framework, the core of the ATLAS Trigger-DAQ (TDAQ) Control system, one of subsystems building the TDAQ system .The key feature of these components is that they are based on the expert-system technology and they should help the TDAQ operator to minimise system down time and to control it smoothly by using the expertise of the system developers.

In the first part we describe the motivation, the design principles, the architecture and some details of the implementation of DVS (Diagnostic and Verification System), Run Control and Setup components. In the second part, DVS component is described in more details including recent developments and examples of its usage for ATLAS commissioning.

DESIGN AND ARCHITECTURE OF RULE-BASED CONTROL COMPONENTS

Motivation and objectives

ATLAS (A Toroidal LHC ApparatuS) Trigger-DAQ system [1] transfers and filters data from the detector front-end electronics to the mass-storage for the offline analysis. It is composed of a big number of hardware and software components, as summarized below:

- 1800 read-out VME boards
- 1800 fiber links
- 150 ROS PCs each hosting 4 ROB-IN cards
- 500 LVL2 PCs
- 90 SFI PCs
- ~2000 EF PCs
- ~30 SFO PCs
- ~50 infrastructure PCs (file servers)
- ~200 Ethernet switches
- and O(10000) applications running on the listed above hardware

The T/DAQ control system shall be capable to smoothly control this number of various components, meeting some requirements [1]. The size of the system under control and the number of components make the probability of a failure very high. The typical failures we are facing every day are PC components failures and applications crashes. Keeping in mind the life-time of the experiment (around 10 years) and the high cost of its down-time and as well the fact that the system will be operated by a non-expert shift crew, it is very important to have system verification, failure diagnostics and recovery facilities embedded in the Control system of the TDAQ. These facilities should allow the Operator to

- Detect problems as early as possible by means of probing the system.
- Make use of the system's developers expertise (knowledge).
- Automate verification of a large system.
- Minimize system down-time using recovery procedures based on problem diagnosis.

Framework approach

Another important aspect of the TDAQ system which makes important contribution to the design, is the geographical decentralization of the experiment and a big number of users all over the world participating in the system development. In order to build the full-scale TDAQ system its software will be used by all ATLAS subdetectors each providing specific functionality, whereas the 'core' software should join all pieces together and guarantee their coherent functioning. This brings the idea of the 'core' framework components providing dedicated services and well-defined interfaces for users and developers.

Design principles

To fulfil the above requirements it was decided to design the system following these principles:

- *Framework* approach: a system shall be configurable and extensible by the experts and users also during the experiment life-time.
- *Expert system* approach: the system's behaviour is described in a rule-based language to allow accumulation of expert's knowledge and an easy adaptation to changing conditions.
- *Hierarchical distributed* architecture of the Run Control system reflecting the tree structure and the scale of the experiment.

Architecture

On the UML diagram below (Figure 1) the actual architecture of the ATLAS Control subsystem is presented.

It is composed of a number of modules or components, implementing particular functionality and interacting with other components via public interfaces. In this paper we

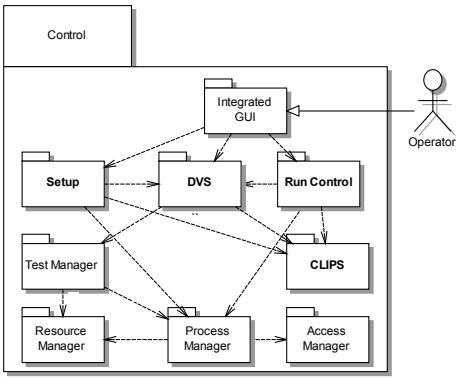

Figure 1. Architecture of the Control subsystem

briefly describe Run Control, DVS and Setup components, all sharing expert system technology provided by the CLIPS package.

Both Setup and Run Control components use verification functionality provided by DVS framework in order to verify the relevant subset of the system under control.

The TDAQ Operator interacts with the system via a human-friendly Integrated Graphical User Interface.

Control subsystem interacts with other T/DAQ subsystems, especially with Configuration and Monitoring ones (not shown on the figure for simplicity).

Run Control

Run Control is a distributed framework allowing each subsystem in TDAQ to define the behaviour of a particular controller and to join all controllers in a distributed tree structure, in order to guarantee synchronous and homogeneous execution of Operator commands through the whole system. The top-level Run Controller represents the status of the system to the Operator and accepts his commands, while leaf controllers deal with applications which directly control the TDAQ hardware. Intermediate controllers represent statuses of different subsets of the T/DAQ.

Each controller is responsible for:

- distributing commands to its children
- receiving children's statuses and analysing (diagnosing) errors
- determining its own state
- undertaking possible recovery actions
- passing status and error information to its parent

The general picture of the Run Control tree is presented in Figure 2.

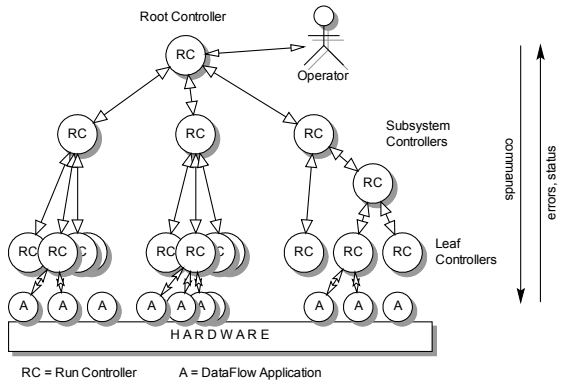

Figure 2. Tree of Run Controllers

Each controller is an application running on some machine in the system. The controller implements a pre-defined Finite State Machine skeleton (common for all controllers), defining its all possible states and transitions; and a small expert system (i.e. an expert system engine and a set of rules or a Knowledge Base, KB) defining his behaviour. The details of the implementation are described in [2]. The default rules included in the framework implementation are very simple, like: *"if all my children controllers are in state 'A', then change my state to 'A'"*, or *"if just one o my children is in 'error' state (e.g. timed-out in transition, reported an error or become unavailable), then change my state to 'error' "*

More sophisticated recovery rules should analyse all available data and make decisions like disabling a sub-tree, executing pre-defined recovery actions or reporting an unrecoverable error to the parent controller, so it could take over the problem.

The concrete recovery policies is to be defined when the final system will be put in operation so the framework should allow easy adaptation of controller's behaviour during experiment run-time.

DVS (Diagnostics and Verification System)

DVS is a framework which allows to:

- Configure a test for any component in the system
- Have a testable view on the particular configuration of a system in a user-friendly GUI
- Automate testing of the system
- Make diagnostics conclusion in case of a problem detected during testing (provided some knowledge put in the Knowledge Base)

More details on DVS are presented in the following section and in [3].

Setup component

Setup component is a 'boot-strap controller' for the initial infrastructure of a particular TDAQ configuration. It brings the system to a state where it can accept RC commands. It uses DVS to verify in depth system's h/w, used by the particular TDAQ configuration used for the current run. At this stage, specific tests developed by particular subsystems are executed in order to detect potential problems and to confirm the system's integrity before launching any process. If some tests fail, user is provided with some options, like *'ignore and continue'*, *'retry'* or *'abort'*. For the final system some automatic recovery rules should be defined.

Setup KB contains additional rules to start, restart and verify applications and diagnose related problems. Most of the infrastructure applications have back-up capabilities, so they are restarted automatically if they crash or even if the host machine goes down. In the latter case a back-up host is chosen and applications restarted there.

The functionality of applications under supervision is also confirmed by the execution of tests.

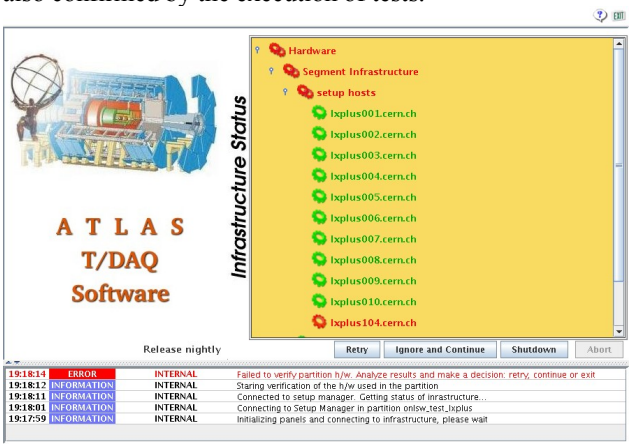

Figure 3. Setup GUI window with hardware infrastructure being tested.

On the Figure 3, the initial GUI window of T/DAQ software is shown. On the right side one can see setup panel with the tree of h/w components (a rack of PCs in this case) being tested, where one PC did not pass test.

CLIPS: an expert system shell

The behaviour of the Run Control components which are mostly data- and event-driven is described naturally in terms of rules, rather then in a procedural language. It is also necessary to have an easy way to update the knowledge about behaviour of the system during experiment lifetime, saving expert knowledge and detailed information about recovery procedures applied by the experts. So it was decided to use expert system technology as the base for all core run control components. Some evaluations ([2]) have been done, and CLIPS expert system shell [4] was selected. It can be shortly characterized in the following:

- CLIPS stands for '**C** '-**L**anguage **I**ntegrated **P**roduction **S**ystem
- Produced by NASA
- Free, open (written in 'C') and well- documented
- Embeddable in other s/w products as a library
- Programming capabilities: rule-base programming paradigm (rules and facts), OO language (classes and objects), conventional procedural constructs

DVS: AN OVERVIEW AND RECENT DEVELOPMENTS

Use Cases

On the Figure 4, the use cases for DVS framework are presented. First, a TDAQ Expert (typically a developer of the system) contributes to the framework by developing specific tests and providing some diagnostic knowledge. Afterwards, TDAQ Operator can reuse Expert's expertise by verifying any subset of the system and diagnosing detected errors. The verification functionality is also used by non-human bodies, like Controllers in order to check the functionality of the subsystem under control.

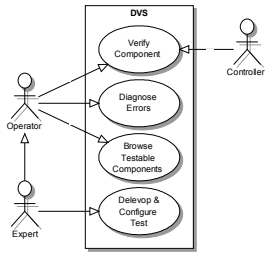

Figure 4 Use Cases for DVS

Architecture

Architecture of the DVS framework is shown on Figure 5. Its core components are the Test Repository database describing all test, and expert system filled with some knowledge by an expert. The functionality is available to end-users via C++ and Java API and via a user-friendly GUI.

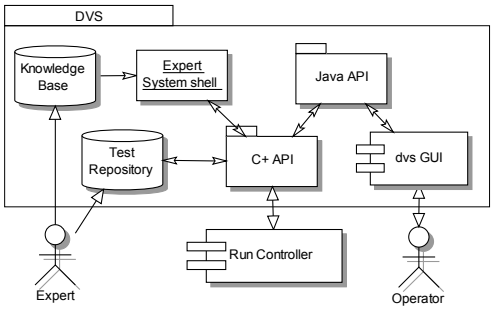

Figure 5 DVS architecture

Tests configuration

A test is a binary running on a particular host in a system. It verifies a particular functionality of a TDAQ component and returns a single result value: PASSED, FAILED, UNRESOLVED, TIMEOUT and some text output. For a single component a number of tests can be associated which can be organized in sequences, executed synchronously or asynchronously. Tests and their relationships are fully described in a database. The following attributes are available for test description:

parameters, host, dependencies, timeouts, scope, complexity and mask. Any test can be associated either to a particular object in the configuration database or to allobjects of a particular type. In the latter case test's parameters and host can be parametrized by attribute values of the concrete objects when the test is launched.

In Figure 6, right side, database editor windows are shown, one with the list of tests defined in the configuration and another one with the parameters of a particular test.

GUI

On the Figure 6 you one can see DVS GUI in action. In its left side the tree of testable components is displayed. User can select a single component or a group in the tree and run all defined tests by clicking a single button. As tests finish, components icons change colour reflecting the result. On the right panel the test output is dumped.

Recent developments

DVS was developed and made available for the end users in 2003 [3]. A number of improvement requests appeared as constant feedback from the users. Many of them were implemented in recent T/DAQ s/w releases:

- Tests *levels* and *masks* for more precise test selection which allow to promptly configure test repository without editing the database. User can select a subset of tests with a particular level of complexity or satisfying some regular expression.
- Asynchronous and synchronous mode for execution of tests for complex objects.
- Test *scope* to prevent conflicting tests from being executed when system is taking data.
- Tests *verbosity* can be defined globally in runtime.
- Test's *runtime output* for long-running tests: user can see output of a test as it executes in real time, even before it finishes.
- Test output produced by all children components can be combined in one panel and then saved in a file.
- New type of interactive tests, called '*actions*', were introduced to allow users to execute more complex test scenarios requiring some input. This also allows to reuse already existing console utilities in the framework. An action is configured as a test, but it is launched in a separate terminal window.

USE OF DVS FOR ATLAS COMMISSIONING

DVS is currently used by different ATLAS subdetectors and T/DAQ subsystems in the ATLAS commissioning activities. A number of tests were developed for different detector and DAQ components, so they can be extensively tested before being installed at the final position. One good example is 'MobiDAQ' system developed by the Tile calorimeter group [5]. Defferent tests for detector front-end electronics and for ROD (Read-Out Drivers) VME modules were developed. On Figure 6 you can see DVS GUI with a number of detector tests being executed.

Another set of tests was developed for ROB-IN PCI modules commissioning.

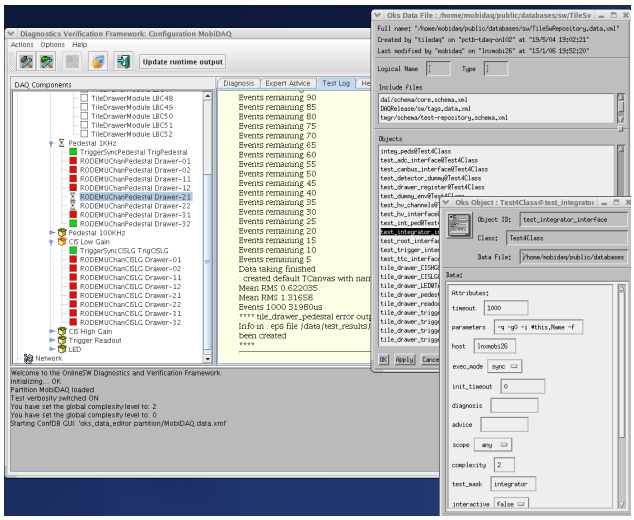

Figure 6. DVS GUI and MobiDAQ test repository in configuration DB.

LARGE-SCALE TESTS

All Control components are a subject for scalability tests, regularly performed on a cluster of PCs. Papers [6] and [7] describes results of the tests in details.

REFERENCES

[1] ATLAS High Level Trigger, Data Acquisition and Control TDR, CERN/LHCC/2003-022 (2003), http://cern.ch/atlas-proj-hltdaqdcs-tdr

[2] D. Liko et all, "Control in the ATLAS TDAQ system", CHEP 2004 Interlaken Switzerland, September 2004

[3] A. Kazarov et all, "Verification and Diagnostics Framework in ATLAS Trigger/DAQ", 2003 Computing in High Energy and Nuclear Physics, La Jolla, Ca, USA, March 2003

[4] CLIPS Expert System Shell, http://www.ghg.net/clips/CLIPS.htm

[5] MobiDAQ web page: http://atlas.web.cern.ch/Atlas/SUB_DETECTORS/TILE/Commissioning/mobidaq/HowTo.htm

[6] ATLAS HLT/DAQ Software Large Scale Tests March 2004, ATL-D-TR-0001, EDMS ID: 499884

[7] D. Burkhart et. all. "Testing on a large scale: Running the Atlas Data Acquisition and High Level Trigger software on 700 pc nodes", CHEP 2006, Feb 2006, Mumbai India.