Verification and Diagnostics Framework in ATLAS Trigger and Data Acquisition

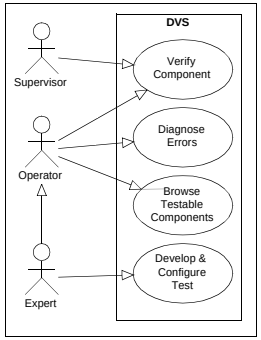
I.Alexandrov[1], A.Amorim[2], E.Badescu[3], M.Barczyk[4], D.Burckhart-Chromek[4], M.Caprini[3], M.Dobson[4], J.Flammer[4], R.Hart[5], R.Jones[4], A.Kazarov[4,6], S.Kolos[4,6],V.Kotov[1], D.Liko[4], L.Lucio[2,4], L.Mapelli[4], M.Mineev[1], L.Pedro[2], Yu.Ryabov[6], I.Soloviev[4,6], H.Wolters[2]

1) Joint Institute for Nuclear Research, Dubna, Russia
2) FCUL (Science University of Lisbon), Lisbon, Portugal
3) Institute of Atomic Physics, Bucharest, Romania4) European Organization for Nuclear Research (CERN), Geneva, Switzerland
5) National Institute for Nuclear Physics and High Energy Physics (NIKHEF), Amsterdam, Netherland
6) Petersburg Nuclear Physics Institute (PNPI), Gatchina, St. Petersburg, Russia

Introduction

Modern Trigger and Data Acquisition systems have a very complex structure and behaviour. Automation of system testing and error diagnosing and recovery are important issues for the control of such systems, because it helps to minimise experiment down-time. In the scope of the ATLAS Trigger and Data Acquisition (TDAQ) software, Diagnostics and Verification System (DVS) was developed. DVS is a framework which allows Trigger-DAQ developers and experts to integrate tests and knowledge into it, so it can be later used by non-experienced shift operator to verify functionality of the TDAQ and diagnose problems.

Use Cases



The main users of DVS are TDAQ Operator, TDAQ Expert, and also TDAQ Supervisor application which helps Operator to control the system.

The following use cases for DVS can be identified:

UC1: TDAQ Expert implements and configures tests for TDAQ components and stores tests in the repository. He also stores the knowledge about testing sequences and components behaviour in DVS Knowledge Base

UC2: During TDAQ initialization, TDAQ Supervisor application or TDAQ Expert launches a number of tests to ensure that h/w and s/w TDAQ components are in a correct state

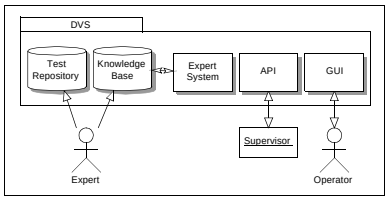
UC3: When an error is detected during the data taking, TDAQ Operator or Expert browses TDAQ configuration in DVS GUI and asks to verify the status of a group of TDAQ components. Using knowledge base and test repository, DVS organizes tests in sequences and execute them, analyses test results, diagnoses errors and presents to the Operator conclusion about the reason of errors and also advises Operator how to repair failed components.

Design approach

The design approach for DVS was

- use *simple* components *tests*, developed by experts, which are stored in tests repository and made available for later use in the framework
- use *expert system* technology to store TDAQ developers knowledge in order to make it available for non-experienced shift operators to help them to control TDAQ

Implementation



For developers (experts), DVS provides an extensible framework which can be used to

- develop and configure tests for classes or objects in TDAQ configuration, or redefine existing ones and store them in test repository
- develop knowledge base, using expert system language, to store specific knowledge about components functionality, which is later presented to the Operator

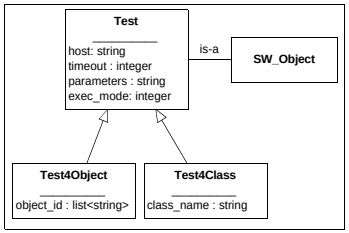
For end users DVS provides

- possibility to have a "testable" view on TDAQ configuration, where user can select a single component or a group of components and verify its status. This functionality is provided via GUI, C++ and Java API.

What is a test

A test is a small application, which verifies status of a single s/w or h/w TDAQ element in a configuration and returns a test result, that can be *Passed*, *Failed* or *Unresolved*. Test should be as independent as possible, i.e. it should not rely on functionality of other TDAQ components. Typically test is developed by a component expert. Test can be launched on any host used in the configuration. It is possible to have a number of tests defined for one TDAQ element. Those tests can be started on different hosts, synchronously or in parallel.

Test Repository



Test repository is a database which allows to describe different attributes of a test and to store all the information in the TDAQ Configuration Database.

Each test in the repository is an instance of one of the three classes defined in Test Repository schema (presented on the right): Test, Test4Object or Test4Class.

The first class allows to define basic test attributes:

- test implementation (as a link to SW_object class from TDAQ Configuration Database schema)
- test parameters
- test timeout
- host name where test shall be executed
- mode of tests execution: synchronous or asynchronous (in case if a number of tests are defined for an object)

Test4Class and Test4Object classes, inherited from Test class used to associate a test to objects in the TDAQ Configuration database. Instances of Test4Object class are tests which verify the functionality of particular TDAQ components, database identifiers of those are stored in '*object_id*' attribute of Test4Object. To define a test for all objects of a particular class, it's necessary to create instance of Test4Class and fill it's '*class_name*' attribute with the name of tested class.

C++ API (Test Data Access Library) is provided to access all the required configuration information.

Currently Online SW test repository contains:

- tests for all TDAQ infrastructure services
- test for computer (remote access test)
- test for VME module ("vme ping" test)
- test for S-Link (source-destination test)

DVS GUI

- implemented in Java
- hypertext navigation
- launched from main TDAQ Control "Integrated GUI" application
- presents TDAQ configuration as an hierarchical tree of testable components, so user con select single component or a group of them
- component's status (component test result) is highlighted
- test output, diagnosis and advice are presented in text panes for each component

CLIPS expert system shell

The core of DVS is an expert system engine, implemented in CLIPS ("C" language Integrated Production System). It's main features are

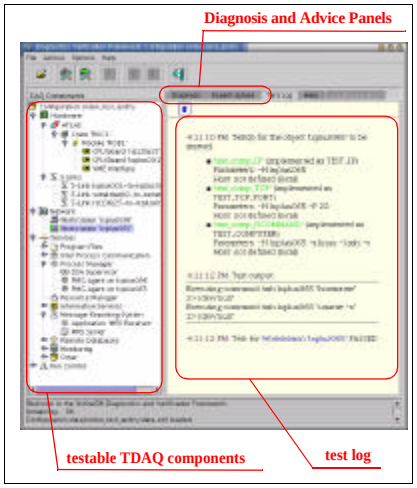
- embeddable in C/C++ applications
- fully featured O-O language + rules
- developed by NASA
- free for non-commercial use
- "de-facto" standard of forward-chaining rule-base (production) system

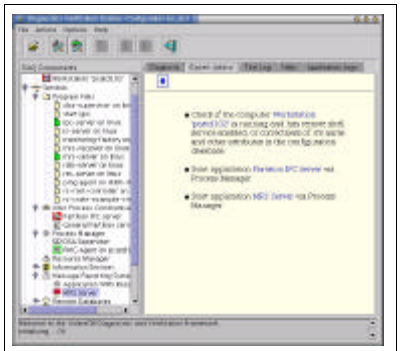
Additional functionality

- Log file browser for accessing log files produced by applications in distributed environment
- Help panel to read on-line documentation for components.

Examples

Screen shot of DVS GUI. There was a sequence of 3 tests started for object "workstation lxplus065". All tests were passed.





Test for application "MRS Server" failed. DVS started a number of additional tests to diagnose the problem. The actions to be done to repair the failed component are presented in "Advice" panel.