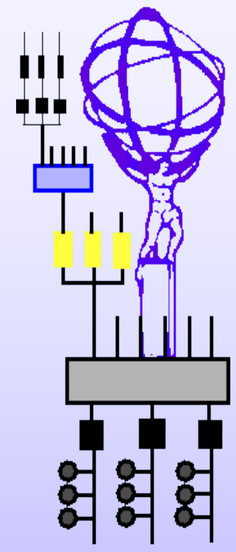
Test Management Framework for the ATLAS Experiment

G. Avolio[3], A. Corso-Radu[1], A. Kazarov[2,3], G. Lehmann Miotto[3], I. Soloviev[1], G. Unel[1]

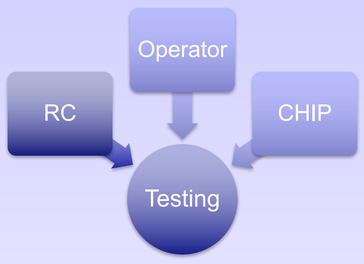
[1]University of California, Irvine, USA
[2]NRC 'Kurchatovsky Institute', PNPI, St. Petersburg, Russian Federation
[3]CERN, Geneva, Switzerland



1. The Online Software of the ATLAS Trigger and Data Acquisition System

The **ATLAS experiment**[1] at the Large Hadron Collider at CERN relies on a complex and highly distributed **Trigger and Data Acquisition**[2] (TDAQ) system to gather and select particle collision data at unprecedented energy and rates. The TDAQ system is composed of a large number of hardware and software components (about 3000 machines and more than 40000 concurrent processes) and is required to handle data coming in parallel from the detector readout over some 1800 point-to-point readout links.

The **Online Software**[3] encompasses the software to configure, control and monitor the TDAQ system. It is based on a number of services which provide essentially the *glue* that holds the various sub-systems together. The **Test Management** is one of these services and is devoted to the verification of the functioning of the TDAQ system by executing tests on request.



The Test Management is based on several components of the Online Software framework. The three most important ones are:

- *Configuration Database (DB):* this provides the description of the TDAQ system (and tests) configuration;
- *Process Manager (PMG):* this offers a service to create, control and monitor the status of all the processes in the TDAQ system.
- *Inter Process Communication (IPC):* this is based on the CORBA communication middleware and allows tests to be invoked on applications as remote procedure calls.

The Test Management is used by:

- *The Run Control (RC) system that periodically verifies the functioning of the components it is in charge of;*
- *The Central Hint and Information Processor (CHIP) that executes tests to diagnose problems;*
- *The operator who manually executes tests via a dedicated graphical user interface.*

2. The Requirements

The Test Management and Diagnostics service[4] was revised after the first data taking period of ATLAS.

A set of **new functional requirements** were added:

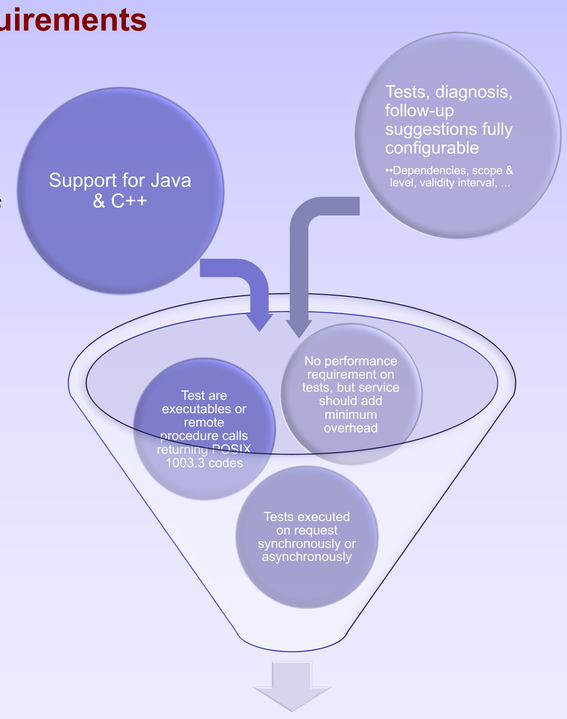
- *"Experts shall be able to define the order in which tests should be executed for a component; the sequence may dynamically change based on the result of completed tests"*
- *"Experts shall be able to define the order with which inter-related components shall be tested; the test sequence may change depending on the result obtained for the components."*
- *"Experts shall be able to define what should be done upon failure of a test or a component to further diagnose the issue or recover."*

All these requirements point towards an increased configurability of the system by ATLAS experts. The description of the testing behavior was thus completely transferred to the configuration database.

An **additional constraint** was added, based on the evolution of other parts of the TDAQ system:

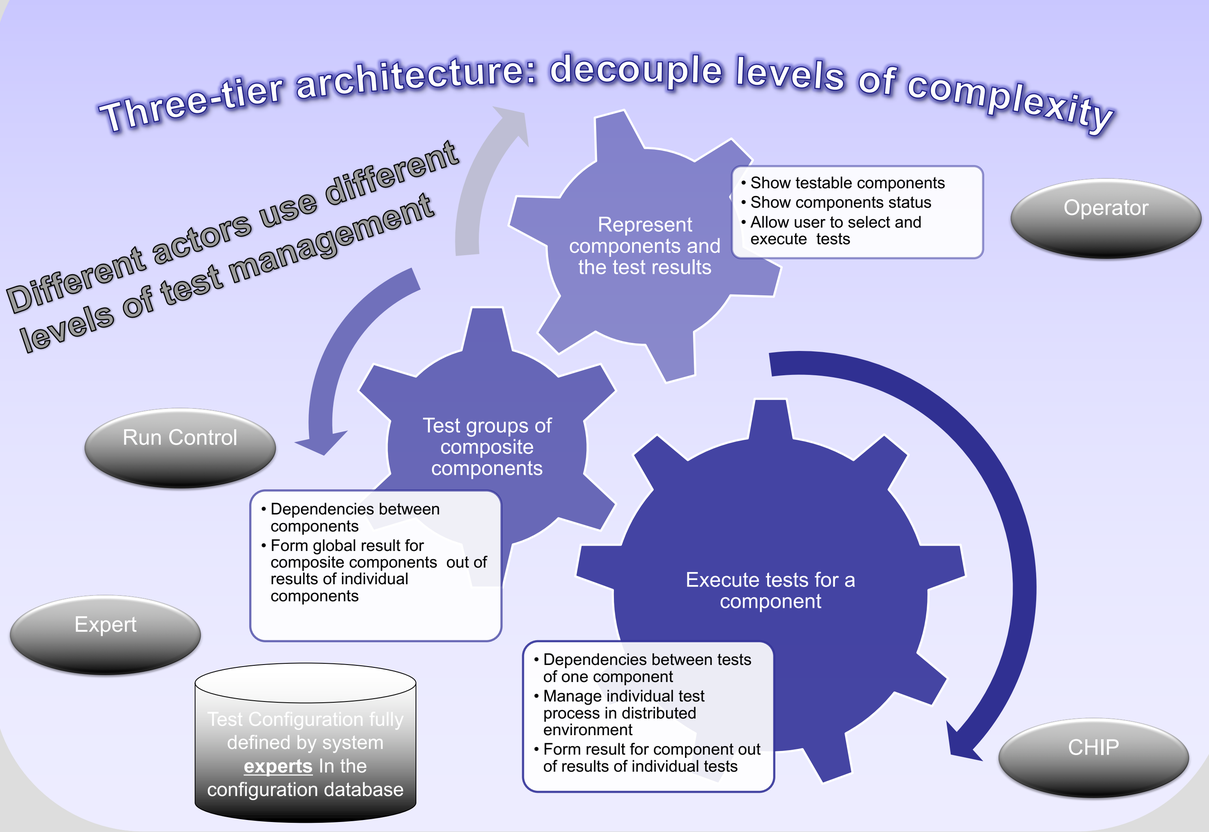
- *"The test management functionality should be provided in C++ and Java."*

Based on this extension of requirements a complete re-implementation has been carried out.

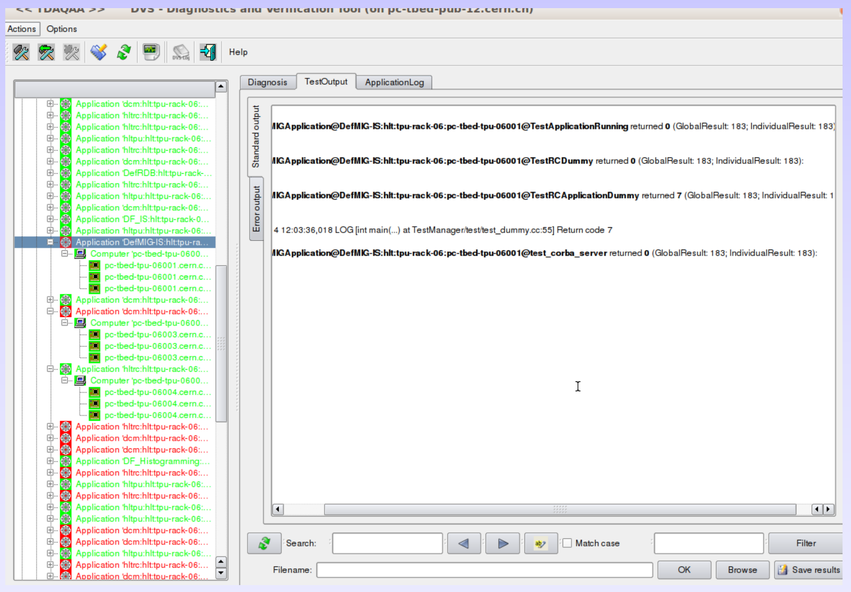


New Test Management Framework

3. Design



4. The Diagnostics and Verification GUI

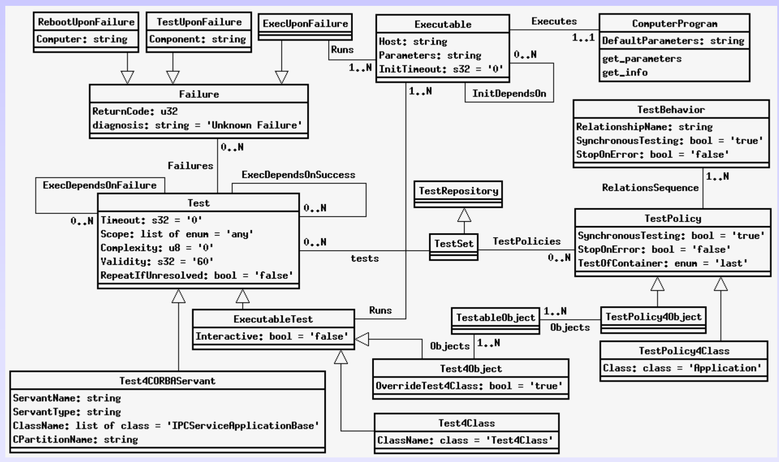


The GUI allows an **operator** to test one or more components, or to run individual tests for one component. The tree-like representation of components indicates their interdependencies (e.g. an application relies on a working computer to run on).
The operator can browse the detailed output of the tests, as well as the diagnosis that has been created based on the test results and the follow-up suggestions that have been configured by the **experts** in the configuration database.

5. Implementation & Performance

The test management framework is modular and is implemented as libraries (Java, C++) and a QT based application for the graphical user interface:

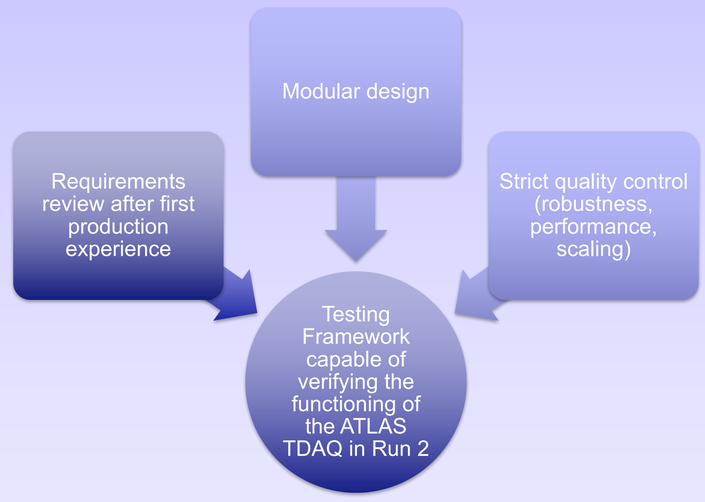
- *A database schema (on the right) to describe tests, components, dependencies and follow-up suggestions with the associated data access libraries to get the configuration information;*
- *A client library to perform tests on individual components;*
- *A client library (C++) to handle the testing of components taking into account the other components they depend on (e.g. before testing an application, test that the computer it should run on works correctly);*
- *A GUI to allow the operator to request the execution of tests.*



The C++ and Java implementations have been developed independently. Both achieve a good, comparable performance (i.e. are capable of launching many tests in parallel without imposing any significant overhead: a test for configuration of 1100 components requires launching of 4500 individual tests which takes about 8 seconds in 32 parallel threads).
The Java implementation is, at the time being, capable of optimally parallelizing execution using less threads than the C++ implementation. Its threading model may thus be ported to the C++ implementation as well in future.

The handling of dependencies between components is implemented at present only in the C++ library. A forward chaining inference engine is used to launch the testing of every component at the appropriate time, based on the results obtained for other components.

6. Conclusions



References

1. The ATLAS Collaboration, 2008, *The ATLAS Experiment at the CERN Large Hadron Collider*, J. Instrum. 3
2. The ATLAS Collaboration, 2002, *ATLAS high-level trigger, data-acquisition and controls: Technical Design Report*
3. Lehmann Miotto G. et al, *Configuration & control of the ATLAS trigger and data acquisition, Nucl.Instrum.Meth. A623 (2010) 549-551*
4. M. Barczyk et al., *Verification and diagnostics framework in ATLAS trigger / DAQ, Nov 22, 2003. 5 pp. Published in eConf C0303241 (2003) TUGP005*