Intelligent operations of the data acquisition system of the ATLAS experiment at LHC

G Anders, G Avolio, G Lehmann Miotto and L Magnoni

CERN, CH-1211 Geneva, Switzerland

Abstract. The ATLAS experiment at the Large Hadron Collider at CERN relies on a complex and highly distributed Trigger and Data Acquisition (TDAQ) system to gather and select particle collision data obtained at unprecedented energy and rates. The Run Control (RC) system is the component steering the data acquisition by starting and stopping processes and by carrying all data-taking elements through well-defined states in a coherent way. Taking into account all the lessons learnt during LHC's Run 1, the RC has been completely re-designed and re-implemented during the LHC Long Shutdown 1 (LS1) phase. As a result of the new design, the RC is assisted by the Central Hint and Information Processor (CHIP) service that can be truly considered its "brain". CHIP is an intelligent system able to supervise the ATLAS data taking, take operational decisions and handle abnormal conditions. In this paper, the design, implementation and performances of the RC/CHIP system will be described. A particular emphasis will be put on the way the RC and CHIP cooperate and on the huge benefits brought by the Complex Event Processing engine. Additionally, some error recovery scenarios will be analysed for which the intervention of human experts is now rendered unnecessary.

E-mail: Giuseppe.Avolio@cern.ch Gabriel.Anders@cern.ch

1. Introduction

The Trigger and Data Acquisition (TDAQ) system [1] of the ATLAS detector [2] at the Large Hadron Collider (LHC) at CERN is composed of a large number of distributed hardware and software components (about 2000 machines and more than 15000 concurrent processes at the end of LHC's Run I) which provide, in a coordinated manner, the data-taking functionality of the overall system.

The Run Control (RC) and the Central Hint and Information Processor (CHIP) are key components of the Online Software framework that encompasses the software to configure, control and monitor the TDAQ system. The RC system steers the data acquisition by starting and stopping processes and by carrying all data-taking elements through well-defined states in a coherent way. Given the size and complexity of the TDAQ system, errors and failures are bound to happen and must be dealt with. The data acquisition system has to recover from these errors promptly and effectively, possibly without the need to stop data taking operations. During LHC Run 1, the detection and handling of problems was based on an embedded rule-based forward-chaining expert system (CLIPS [3]), which was deeply integrated with the RC system. Even though the system performed well, it had major disadvantages: new rules could not be tested without reproducing the error conditions in the production environment and monitoring of system resources used by specific rules was not possible. This made the development and debugging of new rules difficult. Additionally, the expert system was lacking the natural support for detections of temporal patterns.

During the LHC Long Shutdown 1 (LS1) the RC has been completely re-designed and re-implemented in order to address the problems mentioned beforehand. The new RC is now assisted by the CHIP that can be truly considered as its "brain". CHIP is an intelligent application having a global view on the TDAQ system. It supervises the ATLAS data taking, takes operational decisions and handles abnormal conditions. Furthermore, CHIP automates complex procedures and performs advanced recoveries.

2. The Run Control system

2.1. Overview

Applications in the ATLAS TDAQ systems are organized in a tree-like hierarchical structure (the run control tree, see figure 1) following the functional de-composition into systems and sub-systems of the ATLAS detector. Each application in the run control tree is managed by a parent Controller. The root node of the tree is the Root Controller. Controller applications are responsible to keep the system in a coherent state by starting and stopping their child applications and by sending them the proper commands needed to reach a state suitable for data-taking. The composition of the run control tree is completely specified in the TDAQ configuration database [4], where applications and parent/child relationships are described.

Operations across the run control tree are synchronized using Finite State Machine (FSM) principles. FSM transitions are usually initiated by the human operator via a graphical user interface: commands are sent directly to the Root Controller and then automatically propagated throughout the tree of leaf Controllers. Once an application completes the execution of a command (or changes its internal status by any reason) it notifies the parent Controller which in this way can evaluate when a coherent state is reached.

Controller applications are also the RC elements interacting with CHIP. Controllers inform CHIP about any change in the status of their controlled children. CHIP, in its turn, is able to detect any anomaly in the system analysing the status of all the applications and can notify the Controllers about proper actions to be taken. Examples of actions are setting a simple error flag or restarting/ignoring offending applications.

Controllers are able to supervise applications of any kind: "simple" applications not implementing RC functionalities are not expected by their Controllers to follow the FSM but are just started and stopped at the right time (*i.e.,* as described in the configuration database).

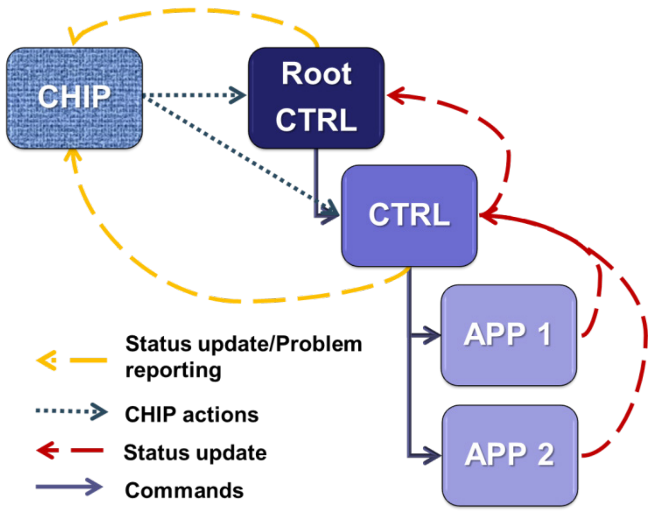

Figure 1. Schema of a simple Run Control tree: the Root Controller (Root CTRL), a child controller (CTRL), two leaf applications (APP 1 and APP 2) and CHIP are shown.

2.2. Communication schema

The communication schema has been designed and implemented starting from the requirement that at any moment in time each Controller needs to know the state of its children, and that CHIP must have a complete and coherent view of the status of all the applications in the system. In such a way Controllers shall be able to determine when a child application has completed the execution of a command or has been properly started/stopped or has changed its state, and CHIP to detect abnormal situations and take recovery actions. As a result:

- any child application or Controller shall be able to notify its parent about any status change;
- any Controller shall be able to notify CHIP about any change in its children's status (the Root Controller being the only Controller sending CHIP information about itself);
- CHIP shall be able to notify any Controller about actions to be taken to resolve an abnormal situation.

Since CHIP, Controllers and child applications run on different hosts of the TDAQ computing farm, they communicate each other via a CORBA based remote Inter Process Communication (IPC) system.

2.3. Architecture of a Controller

The Controller's design is strongly based on the definition of clear responsibilities for all of its components. This strategy allows easy possible future extensions. A Controller is made up of several logical sub-controllers that cooperate together in order to properly deal with incoming commands:

- *Main Controller* – It orchestrates all the operations performed by the Controller; it receives commands from other applications and takes care of their execution delegating, when needed, the responsibility to other components;
- *FSM Controller* – It is responsible for the FSM operations completely encapsulating all the FSM's implementation details;
- *Application Controller* – Its responsibility is to interact with the *ProcessManager* [5] system in order to properly start and stop child applications and monitor its life-time;
- *Command Controller* – It executes commands as requested by the *Main Controller*, keeps track of all the commands currently in execution and enforces that only compatible commands are executed concurrently;
- *DVS Controller* – It interacts with the *DVS* [6] sub-system (implementing the Test Management functionalities) in order to verify the functional status of the controlled child applications.

2.3.1. Commands as resources. The *Command Controller* uses a simple mechanism in order to both keep track of all the commands being executed at any moment in time and to manage the concurrent execution of multiple commands. Commands are treated as if they were resources:

- every command declares a list of other commands it should not be executed concurrently with;
- before every command is executed, the *Command Controller* checks whether some "incompatible" command is already being executed and, if not, a resource for the current command is allocated (the resource being the command itself);
- when a command execution is completed, the corresponding resource is freed.

2.4. Performances

From a performance point of view it is important to keep low the overhead introduced by the RC system in dispatching commands and receiving their acknowledgments. In order to evaluate such an overhead, the time needed by a controller application to fully perform a FSM state transition is measured as a function of the number of child applications. With about one thousand child applications the time needed to perform a state transition is less than 180 ms[1] (see figure 2). Taking into account that transition actions performed by real-life applications during physics runs take tens of seconds and that during the LHC Run 2 a single controller will supervise O(100) children, the controller's performance is fully satisfactory.

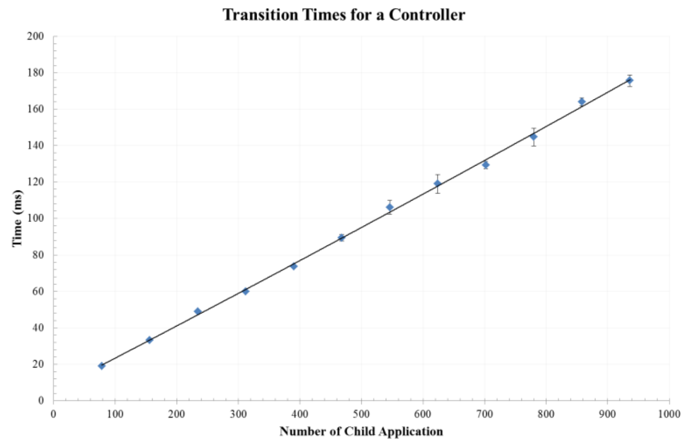

Figure 2. Plot showing the time needed by a Controller to perform a FSM state transition as a function of the number of child applications (evenly distributed on a rack of 39 computing nodes). Child applications are configured to not execute any action during state transitions (*i.e.,* they just receive commands from the parent controller and notify it when the command execution starts or completes).

3. The CHIP

3.1. Overview

The application CHIP gathers various information and employs a Complex Event Processing (CEP) engine in order to aggregate, correlate and analyze this information. Its knowledge base consists of a set of rules loaded from text files. After evaluation of the various CEP engines available on the market, CHIP was based on ESPER [7]. This engine was chosen because of its comprehensive documentation, its open-source license and because it is based on Java which is a well-known and widely used programming language. ESPER allows:

- *Efficient handling of very high information update rates –* The peak rate is typically several tens of thousands of events per second. These peaks coincide with global state transitions of the RC system. Since errors are more likely to occur during state transitions than at other times, it is important that the engine can timely react to failures also in these conditions.
- *Rule testing –* The correct logic of new rules can be verified by artificial injection of events in a unit test. This makes the development of new rules easier, since the behaviour of rules can be tested without creating special conditions in the production environment.
- *Metrics analysis –* It is possible to monitor the CPU usage of individual rules. This is a powerful feature because it allows to identify CPU intensive rules which then can be revised and optimized. In this way potential CPU bottlenecks can be circumvented.
- *Configuration of threading model –* It is possible to change the size of the various engine inherent thread pools in order to tune the engine for best performance.
- *Natural support for temporal correlations –* It is often of high interest to correlate events happening at different points in time. An example for this may be the detection of applications, which, even though automatically restarted after failure, repeatedly crash in a given time window. The frequent crashes may indicate a severe underlying problem and should be investigated.

[1] Tests have been executed on nodes equipped with two Intel Xeon E5645 CPUs, 24 GB of RAM and GbE link connection.

- *Sophisticated anomaly detection* – The ability to perform complex correlations of data from various information providers is one of the main characteristics of CEP engines.

Besides processing the information coming from the RC Controllers, CHIP makes use of the information available in the logging service, the operational data service and the configuration service. For each type of information a corresponding adapter was implemented which makes the information available to the CEP engine (see figure 3). Furthermore CHIP interacts with the Test Management service. Amongst others this allows to detect DAQ problems originating from hardware or network failures.

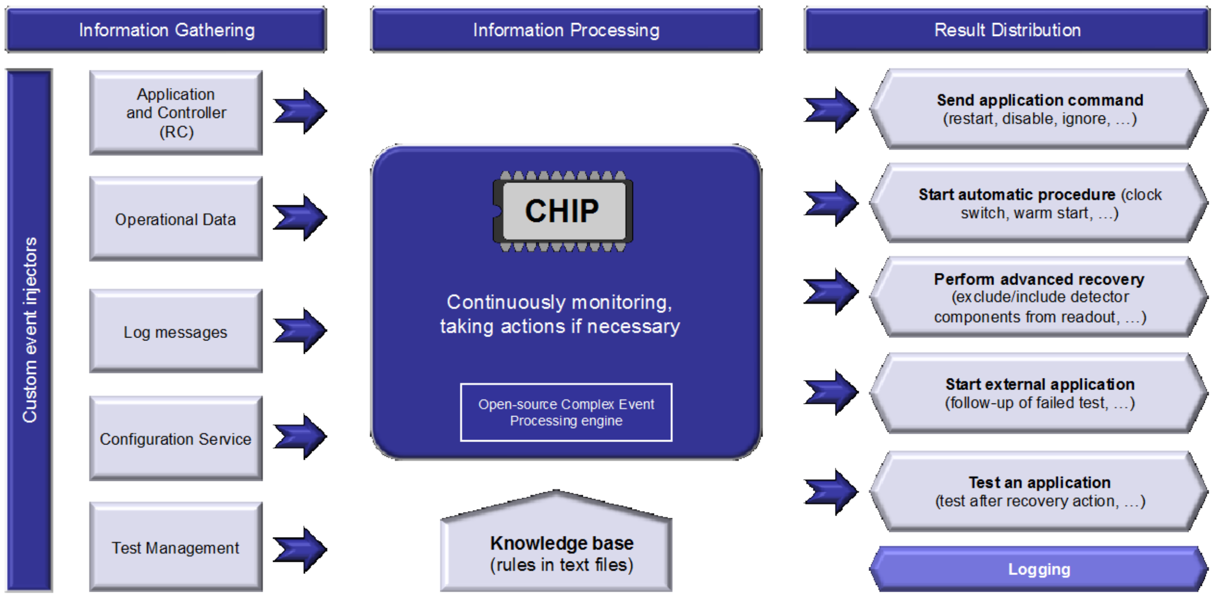
Information Gathering
Information Processing
Result Distribution
Custom event injectors
Application and Controller (RC)
Operational Data
Log messages
Configuration Service
Test Management
CHIP
Continuously monitoring, taking actions if necessary
Open-source Complex Event Processing engine
Knowledge base (rules in text files)
Send application command (restart, disable, ignore, …)
Start automatic procedure (clock switch, warm start, …)
Perform advanced recovery (exclude/include detector components from readout, …)
Start external application (follow-up of failed test, …)
Test an application (test after recovery action, …)
Logging

Figure 3. Architectural overview of CHIP. The flow of information is from left to right and can be divided into three phases: gathering of information using various event injectors, processing of the information within the CEP engine and distribution of results.

3.2. Knowledge base
At present CHIP handles about twenty different kind of recoveries, some of which are very generic, e.g. for crashed applications, others are very sophisticated, e.g. for recovering failing high level trigger applications. Additionally, CHIP automates about ten procedures, e.g. the raise of high voltage of detector components when collisions are established and beam conditions are safe, or the selection of the ATLAS reference clock depending on current detector and beam conditions. The knowledge base is thought of as a non-constant set of rules which is gradually extended over time when new type of errors occur which can be resolved by well-defined recovery procedures.

3.3. Required resources and performance
CHIP runs on a dedicated 16-core machine and uses about 1GB of memory[2]. The CPU utilization follows the incoming event rate and typically varies between <1% and 300% (*i.e.,* the equivalent of 3 CPU cores out of 16). The CPU utilization of the different rules is regularly measured in dedicated test sessions based on the CEP engine's built-in metrics (see figure 4). The responsiveness of CHIP was verified in tests during which the information update peak rate was at least twice as high as expected for LHC Run 2.

[2] Tests have been executed on a node equipped with two Intel Xeon E5540 CPUs, 24 GB of RAM and GbE.

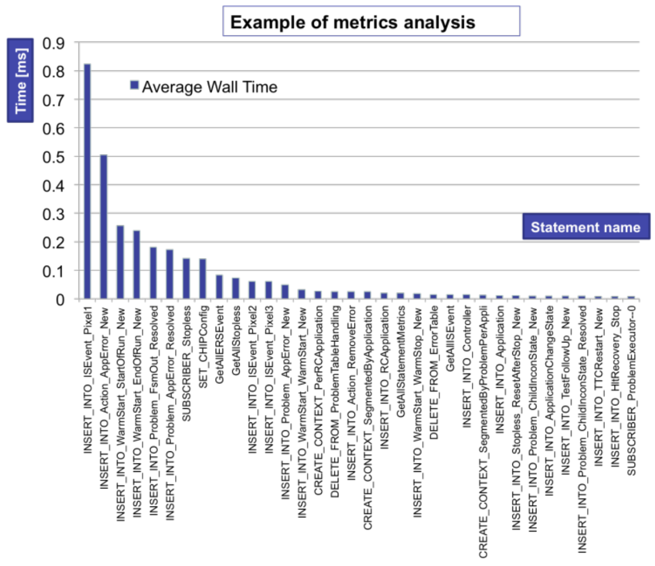

Figure 4. This diagram shows the CPU wall time needed for the evaluation of different ESPER statements (not all existing statements shown). The evaluation time of each statement is averaged over the duration of an approximately 15 minutes long test session, during which the RC was used to cycle the DAQ system through various states and during which various failures were provoked.

3.4. Error recovery example

In case the data taking is blocked due to a faulty sub-detector component, CHIP can dynamically disable the corresponding read-out links without the need for stopping the data taking session. The procedure involves obtaining the read-out link endpoints from the configuration database and communicating with the concerned read-out applications. Once the sub-detector issue has been looked into and resolved, CHIP can re-enable the corresponding read-out components and integrate them back, again without the need of stopping the data taking session. The advantages of this procedure are two-fold: firstly, the recorded data may still be suited for physics analyses depending on the disabled sub-detector channels and secondly, the DAQ system down-time caused by this procedure is much smaller than the down-time caused by a full stop and restart of the complete data taking session. Thus less integrated luminosity is lost for physics analyses.

4. Conclusions

The re-design of the RC system, which is assisted by the application CHIP, achieves the clear separation between the steering and supervising functionality. Whereas the former is taken care of by the RC system, the latter is implemented in CHIP. The decision to base CHIP on a CEP engine allows advanced anomaly detection and error recognition. With unit tests the correct behavior of rules can be verified by injection of artificial events without the need for recreating the error conditions in the production environment. Dedicated test sessions and metrics analyses have shown that CHIP is responsive even during peaks of the incoming information updates. The peak rates during those test sessions were larger than anticipated for LHC Run 2.

References

[1] ATLAS Collaboration ATLAS high-level trigger, data-acquisition and controls : Technical Design Report (Geneva : CERN) https://cds.cern.ch/record/616089?ln=en
[2] ATLAS Collaboration The ATLAS experiment at the CERN Large Hadron Collider JINST **3** (2008) S08003
[3] CLIPS http://clipsrules.sourceforge.net/
[4] Almeida J, Dobson M, Kazarov A, Lehmann Miotto G, Sloper JE, Soloviev I and Torres R The ATLAS DAQ system online configurations database service challenge *Proc. CHEP 2007,* J.Phys.: Conf.Ser. 119 (2008) 022004
[5] Avolio G, Dobson M, Lehmann Miotto G and Wiesmann M The ProcessManager in the

ATLAS DAQ system *Proc. Real-Time Conference, 2007, 15th IEEE-NPSS,* IEEE Trans.Nucl.Sci. 55 (2008) 399-404

[6] Avolio G, Corso-Radu A, Kazarov A, Lehmann Miotto G, Papaevgeniou L, Soloviev I, and Unel G A dynamic test management framework for the ATLAS experiment *ATLAS conference slide ATL-DAQ-SLIDE-2014-226* (https://cds.cern.ch/record/1703297?ln=en)

[7] EsperTECH http://esper.codehaus.org