The ROD Crate DAQ of the ATLAS Data Acquisition System

S. Gameiro, G. Crone, R. Ferrari, D. Francis, B. Gorini, M. Gruwe, M. Joos, G. Lehmann, L. Mapelli, A. Misiejuk, E. Pasqualucci, J. Petersen, R. Spiwoks, L. Tremblet, G. Unel, W. Vandelli, Y. Yasu

ATL-DAQ-CONF-2005-020
20 June 2005

***Abstract*—In the ATLAS experiment at the LHC, the ROD Crate DAQ provides a complete framework to implement data acquisition functionality at the boundary between the detector specific electronics and the common part of the data acquisition system. Based on a plugin mechanism, it allows selecting and using common services (like data output and data monitoring channels) and developing libraries to control, monitor, acquire data and/or emulate detector specific electronics.**

Providing also event building functionality, the ROD Crate DAQ is intended to be the main data acquisition tool for the first phase of detector commissioning.

This paper presents the design, functionality and performance of the ROD Crate DAQ and its usage in the ATLAS data acquisition system and during detector tests.

***Index Terms*—Data acquisition, software design, VMEbus, detector commissioning.**

I. Introduction

In the ATLAS Trigger and Data Acquisition system (TDAQ), the ReadOut Driver (ROD) is a sub-detector specific front-end element [1]. It is located, in the event data flow, after the first level of online event selection, between the front-end electronics and the ReadOut System (ROS), as illustrated in Fig. 1. The ROD receives data from one or more front-end links and sends data over the readout links to the ROS. The ROD Crate DAQ (RCD) was developed due to the sub-detector's need of some common Data Acquisition (DAQ) functionality at the level of the ROD crate, for single or multiple ROD crates, in laboratory setups, at the assembly of detectors, at test beams, and at the ATLAS experiment during commissioning and production.

The ROD system covers all RODs and other functional elements at the same hierarchical level in the event data flow between the front-end electronics and the ROS. Those elements are grouped in crates. The crates contain ROD crate modules which can be: RODs, modules other than RODs, e.g. for controlling the front-end electronics, for driving a Timing, Trigger and Control partition, and one or more ROD crate processors.

In the ATLAS implementation, ROD crates are custom 9U VMEbus crates, each one housing a single board computer running Linux, acting as the ROD Crate Controller (RCC).

RCD comprises all the software to operate one or more ROD crates. It provides the functionality for configuration and control, data readout, ROD emulation, monitoring, and event building across multiple ROD crates.

RCD is based on existing ROS software originally developed to meet the requirements of the main dataflow at the level of the ROS: to buffer fragments which are input via standard ATLAS readout links and provide fragments on request from the Level2 trigger and Event Building systems. An analysis of the ROS framework indicated that this could be generalised to the detector specific level of the ROD crates thereby extending the domain of ATLAS DAQ for which common software support is provided.

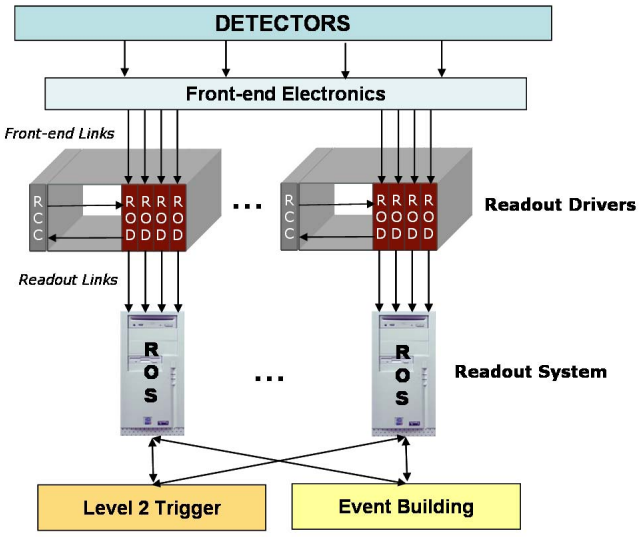

Fig. 1. Simplified diagram of the TDAQ chain.

Manuscript received June 17, 2005.

S. Gameiro, D. Francis, B. Gorini, M. Joos, G. Lehmann, L. Mapelli, J. Petersen, R. Spiwoks and L. Tremblet are with CERN, Geneva, Switzerland.

G. Crone is with the Department of Physics & Astronomy, University College London, Gower Street, London WC1E 6BT, UK.

R. Ferrari and W. Vandelli are with the Università di Pavia and I.N.F.N., Via A. Bassi 6, IT-27100 Pavia, Italy.

M. Gruwe was with CERN, Geneva, Switzerland, She is now with the Gesellschaft für Schwerionenforschung mbH, Planckstraße 1, D-64291 Darmstadt, Germany.

A. Misiejuk is with Royal Holloway, University of London, Department of Physics, Egham Hill, UK-Egham, Surrey TW20 0EX, UK.

E. Pasqualucci is with the Universitá di Roma I 'La Sapienza' and I.N.F.N., Piazzale Aldo Moro 2, IT-00185 Roma, Italy.

G. Unel is with the University of California, Irvine, Department of Physics & Astronomy, Irvine - CA 92697-4575, USA.

Y. Yasu is with the High Energy Accelerator Research Organization, KEK, 1-1 Oho, Tsukuba-shi, JP-Ibaraki 305-0801, Japan.

II. RCD Use Cases

This chapter describes the RCD use cases [2].

A. Module Control

The RCD is used to control one or more modules in a ROD crate. Controlling a module consists in communicating with it, passing relevant information at each of the TDAQ system state transitions (see section III.C). An example of modules that use RCD only for controlling are trigger modules. All the other use cases presented in this chapter include the RCD control functionality.

B. ROD Emulation

In this case, the RCD is used to read non formatted data from one or several modules, and to build ROD fragments, which can be later sent to the TDAQ system. This is typically used when a detector ROD is not available, and allows testing a detector with simpler electronics. Fig. 2 illustrates this use case.

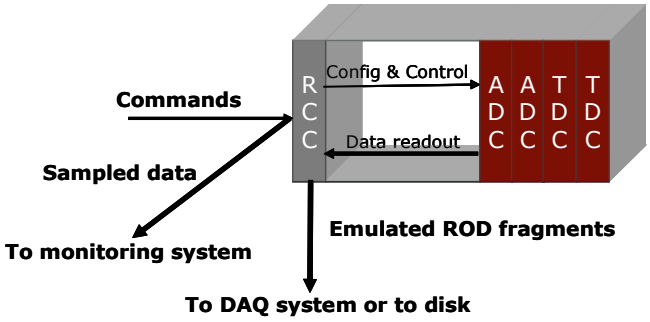

Fig. 2. ROD Emulation use case.

C. ROD Data Readout

This is the main use case for the Phase 1 of detector commissioning, when the RCD is the DAQ tool available for detector readout. The RCD reads ROD fragments from one or several ROD modules via VMEbus, and builds ROS fragments. These ROS fragments can be either stored locally or sent out via a TCP connection to a data driven event builder, as described in section II.E.

D. ROD Monitoring

This is the typical use case for the RCD during ATLAS runs. The ROD modules are fully controlled by the application, but the modules themselves manage the primary data flow, sending events to a ROS through a readout link, as is show in Fig. 3. In this case, the RCD acquires sampled fragments from RODs for monitoring purposes, sending them to the monitoring system or the histograming service. The RCD should also be able to detect errors signalled by the ROD modules and perform operational monitoring.

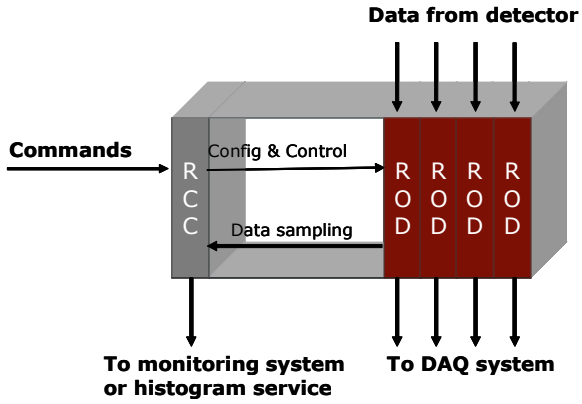

Fig. 3. ROD Monitoring use case.

E. Data Driven Event Building

When acting as data driven event builder, the RCD receives ROS fragments coming from VMEbus crates or ROSs via TCP connections, and builds full events (Fig. 4). This is mostly needed during the detector commissioning, when the full TDAQ system and in particular the high level trigger is not yet available.

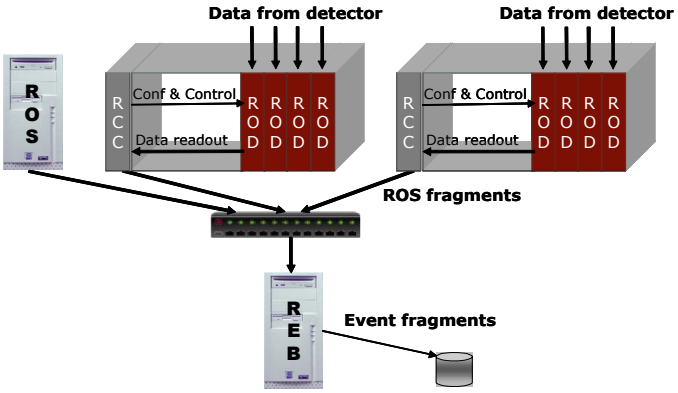

Fig. 4. Data driven event builder use case.

III. RCD Architecture

The RCD software is based on a C++ common application framework originally developed for the ROS and referred to as IOManager (IOM). The framework loads specific plugins to customize its behaviour for the different RCD deployment scenarios.

This chapter describes the RCD architecture, focusing on the thread and plugin structure, and the interface with the TDAQ run control system.

A. Threads structure

The RCD has the structure of a single, multi-threaded process based on Linux POSIX threads as illustrated in Fig. 5. The trigger thread is activated on the occurrence of an event which may be external or internal to the RCD. It builds a request which describes actions to be performed by the RCD in response to the event, e.g. reading out a number of data sources, and then writes this request to a queue. A request handler thread processes the requests: reads an element from

the request queue and executes it. It typically consists in extracting fragments from the data sources, and assembling these into a larger fragment, which is written to an output device. Several request handlers may work in parallel thereby achieving a better CPU utilisation.

The framework provides a number of optional threads. The S*equential Input Handler* thread supervises the input of data fragments from several ROD modules, based either on polling of registers in the RODs or VMEbus interrupts. The data fragments are read into a set of internal memory buffers, one per ROD, from where they can be accessed (randomly) via the request handlers. The *User Action Scheduler* thread allows activation of user written handlers when a defined time slice (per handler) has expired. Similarly, when a VMEbus interrupt occurs, the *Interrupt Handler* thread activates a user written handler identified by a VMEbus interrupt vector.

In addition to the threads related to the data flow, there are control threads which communicate with external controller applications for configuration, error handling and statistics.

The strategy for thread scheduling is based on 'poll and yield': a thread polls on a resource, executes until the resource is exhausted and then yields, i.e. gives the CPU back to the OS (Linux). This approach minimises the number of thread context switches and the time for one context switch.

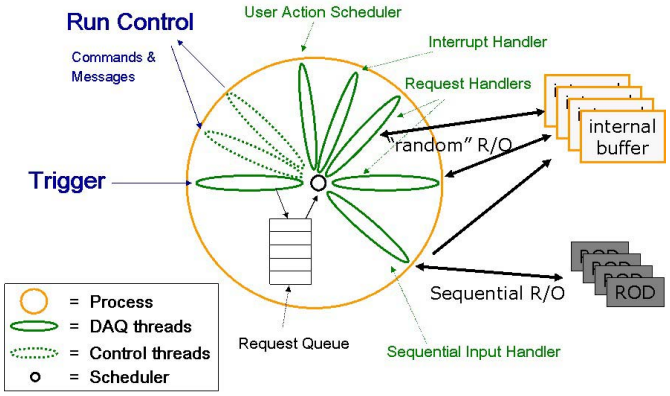

Fig. 5. Thread diagram of the RCD application.

B. Plugins Mechanism

The application framework implements the common core functionality of the system: interface with the run control commands, error handling and recovery, activity scheduling, configuration management, and event selection and sampling, among others. The application relies on specific plugins for the implementation of the different I/O protocols.

All the plugins are loaded by the application at run time. They are loaded dynamically, and do not need to be linked to the main application. This allows users to add new plugins without having to modify the application binary.

Fig. 6 shows a diagram with the four different types of plugins: configuration, trigger, module and output. The configuration plugin is the first one to be loaded, and it informs the application about the other plugins. For each of the four types of plugins, an abstract base class defines an API that needs to be implemented for every specific instance. A more detailed description of each plugin is now presented.

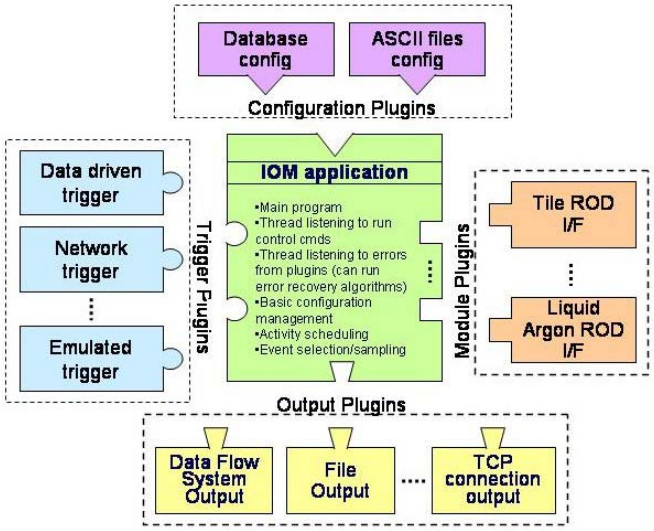

Fig. 6. Plugin diagram of the IOM application.

1) Configuration

The standard way of configuring the RCD is via a configuration database. This database is based on a core schema which includes the definitions of all the classes of the system. A modular structure allows creation of new sub-schemas with user defined classes, which may use other classes from the core schema, and can be included in the user's database. An alternative way of configuring the RCD is via simple ASCII files, more suitable for development/debugging phases.

2) Trigger

The trigger plugin implements the functionality of the trigger thread, see section III.A. The RCD software provides several trigger plugins characterised by the type of trigger source which may be external or internal to the application: arrival of a message on a network; input of data fragments in the internal buffer via sequential data channels, see Fig. 5 (data driven trigger); internal generation of the trigger (emulation). In addition the user may develop trigger plugins driven by detector specific trigger hardware, e.g. VMEbus modules.

3) Module

A module describes a hardware or software component that is controlled by the framework. It can perform any appropriate action associated with a state transition; the base class associated with a module is called ReadoutModule, and it includes virtual methods for each of the state transitions. Typical actions include reading the configuration database, initializing the hardware modules or publishing statistics.

A module may be associated with the readout of data. In that case, one or more data channels are defined within the module, with virtual methods for requesting, retrieving and deleting fragments. A specialised implementation of a data channel was provided for RCD deployments, which include virtual methods for polling on a data source and retrieving

data (ROD fragments), sequentially.

4) Output

As explained in section III.A, a request handler writes the event fragments to an output device. This functionality is implemented via an output plugin. The RCD software provides several plugins associated with the type of output device e.g. network (DataFlow System Ouput) or local disk. In addition the user may develop specific output plugins although this should be an exception.

C. Supervision System

The TDAQ Supervision system is in charge of performing the initialization and shutdown of TDAQ firmware and software, distributing commands to TDAQ elements and synchronizing operations between them, and performing error handling [3]. The building block of the Supervision is the *Controller*. The Supervision will generally contain a number of Controllers organized in a hierarchical tree.

In order to regulate the control activity, a State Machine model was introduced in the Controller's core: a Controller always has a state, reflecting the possible states of the TDAQ system. The states and transitions of all Controllers in the tree are synchronized. Each state has a defined set of authorized transitions that bring the system into a new state, as shows Fig. 7.

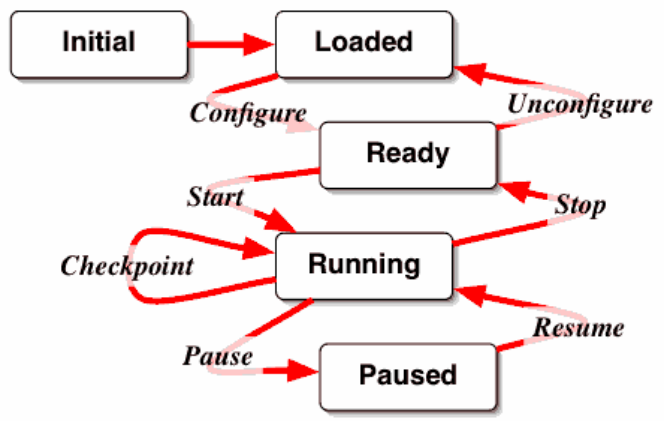

Fig. 7. Simplified diagram of the Controller's State Transitions.

IV. RCD Deployment

This chapter deals with relevant aspects of recent deployments of RCD: its usage during the 2004 Combined Test Beam, enhancements of the functionality and its role in the detector commissioning phase.

A. 2004 Combined Test Beam

In the 2004 Combined Test Beam, a slice of the ATLAS detector was assembled with the objective of integrating and testing all the components of the experiment under real conditions. Elements from nine different types of detectors, including the inner detector, calorimeter and external muon chambers, were installed in a beam line and exposed to different kinds of particle beams.

The detector readout electronics was connected to a scaled-down version of the final ATLAS DAQ system, which was operated in realistic data taking conditions (24 hours a day 7 days a week) for about six months.

The readout electronics for the various detectors was mostly available in early prototyping versions, sometimes with limited functionality; hence the RCD software had to be used for different tasks:

- All the detectors but one *controlled* and *configured* their modules via the RCD.
- Some detectors used the *ROD emulation* functionality to read non-formatted data from different types of modules, combine it and build formatted fragments, as the final ROD modules were not available at the time.
- More than half of the detectors had real ROD modules and some performed *data readout* with the RCD.
- All the detectors but one used the RCD *monitoring* functionality.

RCD was also used to control and configure the three different VMEbus modules used to interface the readout electronics with the first-level trigger, to handle incoming timing and trigger signals and to provide feedback on busy conditions.

B. RCD enhancements

After the 2004 Combined Test Beam, the usage of the RCD was analysed in detail, and a number of changes were performed in order to simplify the API and add some functionalities [4].

To test the new software, and while the final RODs of the detectors were not available, a test bed was set up to emulate the behaviour of a ROD. It consists of a VMEbus memory module plus a VMEbus interrupt module [5], which reproduce the three main ROD functionalities: memory, registers and interrupt capability. The memory is filled with ROD fragments to emulate the ROD data source. The trigger to read the ROD fragments is provided by the VMEbus interrupt module, which can be used in either polled or interrupt driven mode.

When doing polled data read out, a register in the VMEbus interrupt module is read for the presence of a "data available" bit. This bit is set via a NIM pulse which runs at a given frequency. When the bit is set, a fragment is read from the memory. When using interrupts, every time a fragment is available (given again by a NIM pulse), the VMEbus interrupt module issues an interrupt, and a fragment is retrieved from the memory.

Fig. 8 shows the performance measurements done for such a test system running with two readout modules, each one of them with two data channels. The measurements were performed for both polling and interrupt modes, to compare the event overhead and in general the ROD fragment transfer time.

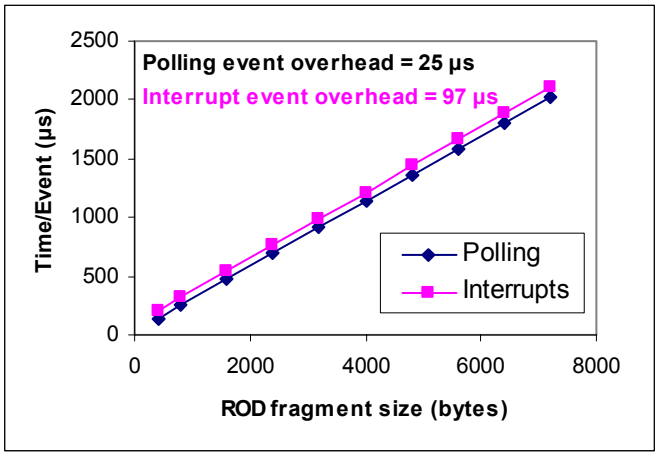

Fig. 8. Performance measurements of a ROD crate test system running four data channels.

The total transfer rate measured in this system, using single cycles to read from the VMEbus memory, is 3.6 Mbytes/s, which is close to the maximum rate that can be achieved in this system.

It can be observed that the event overhead is smaller than 100 µs, corresponding to an event rate of 10 kHz, which fulfils the RCD requirements.

One of the enhancements to the RCD framework was the capability of acting as an event builder (see section II.E and IV.C). To validate the new event building software, a PC running a RCD event builder was added to the test system described in section IV.B. Full events are built out of the ROS fragments received from the VMEbus crate. The network link between the ROD crate and the RCD event builder is a Fast Ethernet connection (100 Mbit/s). The performance measurements can be viewed in Fig. 9.

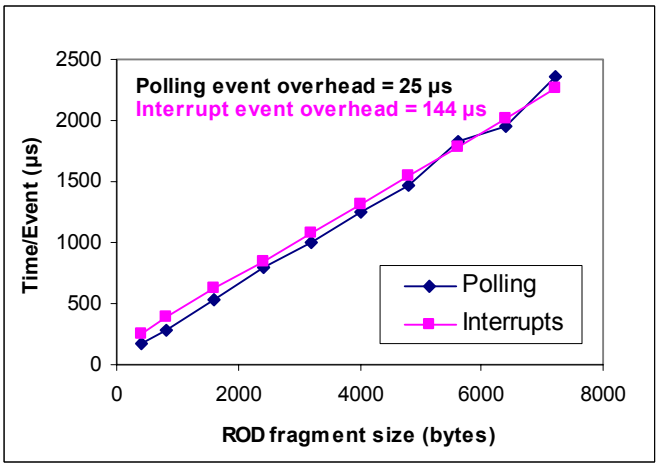

Fig. 9. Performance measurements with a ROD Emulator connected to a RCD event building system.

The performance is comparable to that shown in Fig. 8, which shows that adding a RCD event builder has a small impact on the performance of the system.

C. *Detector Commissioning*

The commissioning of detectors in ATLAS has already started and will last until 2007. Within this context, RCD will be the main tool concerning the data acquisition functionality.

The usage of the RCD will be similar to that of last year's test beam, with the ROD emulation modules gradually being replaced by the final detector RODs.

In addition, whenever a larger part of a detector or several detectors will have to be readout, requiring more than one ROD crate, multi-crate event building will be used to acquire combined data.

V. Conclusion

The ROS software framework, originally developed to meet the dataflow requirements at the level of the common readout system in the ATLAS DAQ (see Fig. 1) , has been extended to the detector specific domain, largely based on VMEbus, between the front-end electronics and the readout system. Due to the modular design of the ROS software, and in particular the technique of using plugins, new functionalities could be added so as to meet the additional requirements of RCD.

A first version of RCD was tested in the ATLAS Combined Test Beam of 2004 and the requirements were met both in terms of functionality and performance. Based on the feedback from the test beam, RCD was further developed mainly to add the event building capability and to improve the handling of VMEbus modules. The RCD software will be an important data acquisition tool for use in testbeds, for detector commissioning and as a component of the final ATLAS DAQ.

Acknowledgment

The authors would like to thank the ATLAS TDAQ community for their contributions in the work presented in this paper.

References

[1] ROD Crate DAQ Task Force. "Data Acquisition for the ATLAS Read-out Driver Crate (ROD Crate DAQ) - Definition –" [Online]. Available: https://edms.cern.ch/document/344713/1

[2] G.Crone, S.Gameiro, B.Gorini, M.Joos, E.Pasqualucci, J.Petersen. "ROD Crate DAQ User's Guide" [Online]. Available: https://edms.cern.ch/document/577958/1

[3] A. Kazarov, G. Lehmann, D. Liko, S. Wheeler, H. Zobernig. "ATLAS TDAQ: Controller Requirements". [Online]. Available: https://edms.cern.ch/document/431663/2.1

[4] G.Crone, B.Gorini, M.Joos, E.Pasqualucci, J.Petersen., W. Vandeli. "Extensions of the IOManager architecture for ROD Crate DAQ". [Online]. Available: https://edms.cern.ch/document/554806/1

[5] Creative Electronic Systems S.A. "RCB 8047 CORBO VME Read-Out Control Board, User's Manual". Switzerland, Aug. 1993.