Experience Using CMS Daq Column System At H2 Test Beam Area

M. Bellato[1]*, L. Berti*[2]*, M. Gulmini*[2]*, G. Maron*[2]*, R. Ponchia*[2]*, N. Toniolo*[2]*, G. Vedovato*[2]*, S. Ventura*[1]*, X. Yang*[2]

[1] INFN – Sezione di Padova, Via Marzolo 8, 35131 Padova, Italy
[2] INFN – Laboratorio di Legnaro, Via Romea 4, 35020 Legnaro, Italy

Abstract

While achieving the task of providing a daq system for the CMS muon chamber testbeam held at Cern on july '99, we took the opportunity to integrate a complete "CMS daq column" prototype for the first time on a real data taking environment. Experience was acquired on the customization of the generic event builder for a specific setup and the whole of the functionalities and protocols needed to drive such a multisource architecture were put under validation. As a result we could spot many missing and inadeguaties of the system implementation on the different aspects of configuration, control and synchronization. Based on the obtained resuts, a major review of the whole system is under way, aiming to the specification of a daq tool which could become a baseline for the future testbeam setup, moreover in sight of a multiple subdetector readout integration.

keywords daq, readout, event builder, test beam

1. Introduction

During last year we were involved in a major revision of the daq system for the Padova CMS muon chamber[1], meant to provide chamber production validation and to support data acquisition on next tests on the beam of the chamber prototypes. We then started evaluating the availability of many software components developed to investigate and demonstrate interconnection technologies for the final CMS daq system, and decided to setup a "daq column" (front end control, network interconnection, processing and storage workstation) based on such components. Although so far only used with dummy data, those already provided all the underlying structure needed to synchronize data movement among several sources to different destinations and were already available on various platforms (both hardware and software speaking). Thus our actual task had been to adapt those generic data movers to our specific input (readout hardware) and output (database or mass storage) integrating the synchronization with the trigger system.

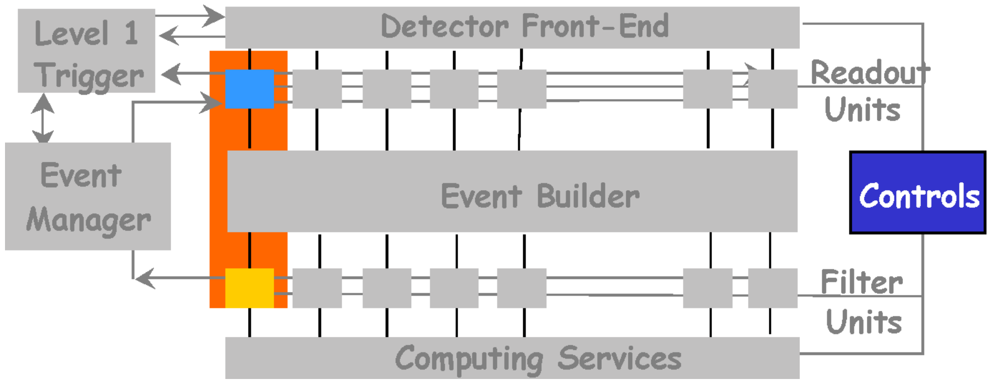

Figure 1: The CMS Data Acquisition system tructure

The first working setup was run during July 99, when one chamber sample has been under test for two weeks on the H2 beam area at CERN. Readout requirements were quite relaxed in terms of performance (64 TDC channels, typ. event size ≈ 100-500 bytes, typ. trigger rate ≈ 800-8k events/spill), but our system enhanced the previous daq's with the integration of the silicon telescope readout and adding direct connection to the OO framework (ORCA[2]) for the data storage. Besides physics measurements, the main goals of this run were to evaluate the actual needed effort for the customization of the generic components, to validate on a real condition event building models and protocols and, eventually, to propose a general tool for future testbeam needs in the CMS collaboration.

2. The hardware setup

As depicted in fig.2 the installed system included two VME based frontend's controlled by Motorola 200 MHz PowerPC boards running vxWorks RTOS: the fist crate hosted the chamber readout hardware (TDC's, BTI, PU, Delay) while the second was interconnected via a National Instruments MXI link to the H2 old daq crates, through which it could snoop the dataflow from the Silicon Telescope device.

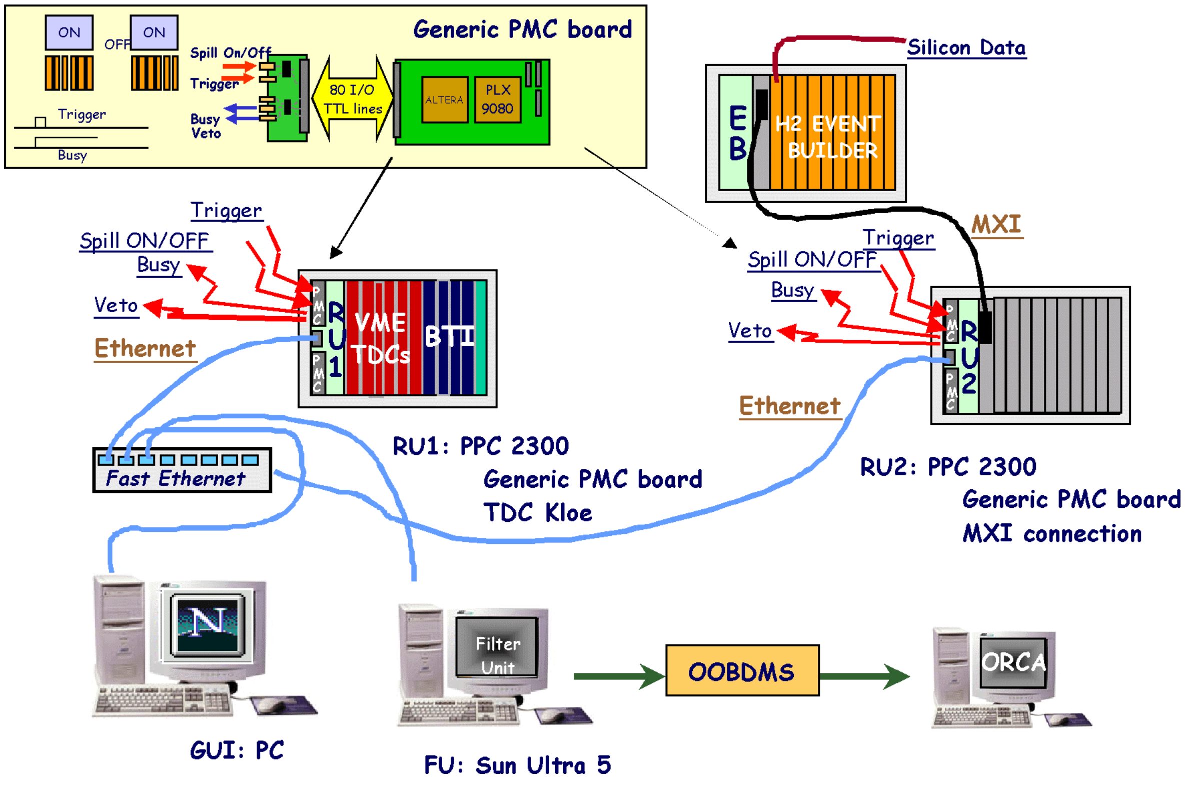

Figure 2: Hardware setup installed at H2

An Ultra 5 Sun workstation was providing the data collection and database filling and a second workstation the Web based run control. The subsytems were interconnected through a dedicated 100 Mbs ethernat switched network, while the first level trigger distribution was handled with a pair of generic customizable I/O PMC moduiles (ref.) featuring two interrupt sources (first level trigger and beam spill) and two synchronization lines (system busy and trigger veto).

3. Daq Software Architecture

3.1. The Readout

The class model as implemented on the CMS daq software package^3 we were using has a configurable multi-component architecture over a toolbox library taking into account the communication issues among the different elemnts of the daq system (fig.3).

Every component brings a specific dataflow (trigger input, f.e. control and data reading.for the RUI, local buffering for the RUM, data expedition and higher level triggers for the RUO) to a clean stream based interface. The resulting “daq mainloop” code shows as follows:

```
for (;;) {
        try {
      // Waiting trigger
            *ruiTrgStream >> setl(sizeof(trigger)) >> (char*)&TBtrg;
      //Read Event
            *ruiInputStream >> setl(1) >> (char *)evt_data;
      //Write to RUM memory
            rumStream_->open(&event,vxios::write);
            *rumStream_ << setl(evt_data[0]*sizeof(int))
                  << (char *)evt_data;
            rumStream_->close();
      }
}
```

Our actual customization of the generic framework has been indeed to develop the specialized classes to interact with our hardware and perform the correct data collection procedure.

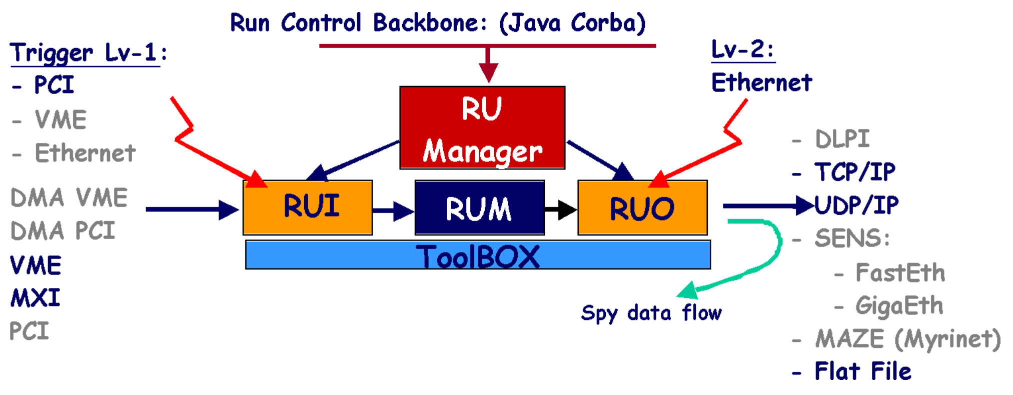

Figure 3 : Software components of the readout unit

At boot time specific components are instantiated to fit with the actual configuration: in our setup we had two flavors of RUI (VME frontend accesses on first crate, MXI control and data snooping synchronization on the other), and three flavors of RUO (TCP data sending, UDP data snooping, and direct storage on local disk).

The top rates and throughput obtained by such setup were measured at 100 kHz of trigger handling for the RUI, and from 6 to 9 kHz for the whole RUI-RUM-RUO dealing with events sized respectively 4kB and 256 bytes.

3.2. Event Building

A similar analogous multi-component structure was driving the collection of the event data on the receiving side of the daq column (fig.4). The FUI was broadcasting event requests and handling the receiving of the data, with event fragments merged on the FUM (a multibuffer memory) and then forwarded to the FUO to be stored. On that side of the column most of our customization was done to develop a new UDP/TCP based FUI and on the interfacing to the OO database populator, which had to be optimized and adapted to build our specific event objects. A spy FUO was also added to monitor dataflow.

Differently to the daq column demonstrator, we actually decided to avoid a separate Event Manager task. Dedicated hardware logic guaranteed the first level trigger consistency on the two RU's with an appropriate backpressure.

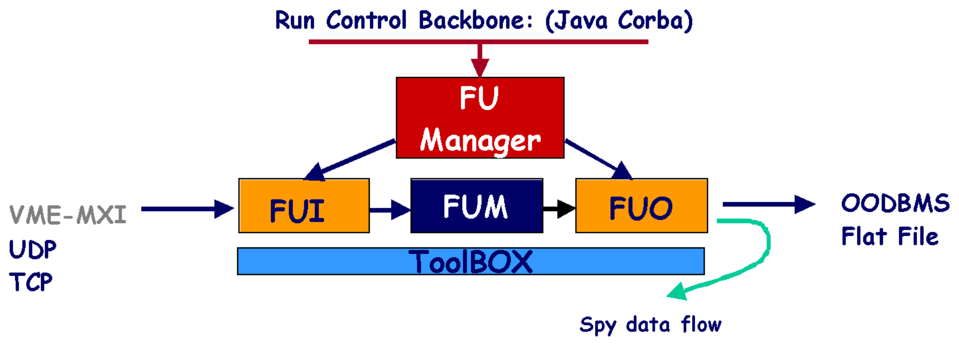

Figure 4 : Software components of the building unit

Due to an internal pipeline on the existing Silicon readout system, the merging of events had to be postponed after the end of beam spill. All of the events collected during the spill were thus aggregated on a single "super event" by each RU, and the super events were sequentially requested by the FU's. This modification involved a change on the building protocol and on the RUI trigger handling as well.

The whole RU+FU chain was tested at effective rates of up to about 1 kHz , although this wasn't needed for H2 super events (bean spill occurs every 14.2 s).

3.3. The Run Control System

Configuration, control and monitor of the whole daq system was relying on a Java-Corba based system developed by ourselves[4]. Every component of the daq has its own manager (a specialized customizable state machine) hierarchically connected on a Corba backbone through which command dispatching and status probing is made. The same backbone hosts some service managers (a database for the configuration, a logger and the user interface).

Aiming to the maximum portability, managers were developed in Java. Nevertheless the huge footprint of the virtual machine obliged the removal of the manager code from the RU's: with the manager running on a remote machine and a proprietary tcp/ip protocol then bridging the JNI calls to the given VME board.

A Javascript library provides a complete environment to control the system. It also features dynamic html generation needed to achieve web-based GUI's.

4. System Evaluation

As stated in tab.1 our setup finally covered the daq needs at H2 and produced real data which could be displayed and analyzed within the ORCA environment.

Finally the whole setting up of such system required nearly six man months of manpower, not very far to what would have been needed starting from scratch. Besides the usual debugging needed to start a new system, we encountered few major flaws. They mostly regarded the user interface (both on event display and run control) which slowed the debugging itself, and a lack of flexibility in the RU and FU components models which required to dig into class code to implement the needed trigger handling and synchronization. This last issue has already been addressed by a major revision of the whole daq toolbox, based on remote method invocation software model, which results in a reduced dependence among the single components of the daq system.

Performances		
Total throughput wasn't a big issue (10-100 kB/s) due to spill cycle.		
Level-1 trigger handling within requirements (> 500 hz)		
Uptime		
60% of the two weeks run (on different RU configurations). Half of the runs were taken only on flat file storage.		
Required Manpower	this setup	to a new front end
customization	3 man months	≈ 10 days
integration	2 man months	
final setup debugging	3 man weeks	probably same

Table 1: Summary of the daq column integration

This said, the integration of a new frontend to this system has been demonstrated to be very effective (less than two weeks for the silicon telescope) if it relies on the same event building model. Moreover the tight compliance to the CMS daq demonstrators API guarantees scalability of such systems and will allow easier upgrades to new technologies or building protocols as soon as new demonstrators will be available.

As a next step, the renewal of the H2 general daq system facility is now underway, and will be based on the same software and hardware framework: incoming users will just need to provide a specialized RUI and adapted database populator in order to hook to the system.

References

[1] M. Aguilar-Benitez et al.,"Performance and Mechanical tolerances achieved with a full size prototype of a CMS Barrel Muon Drift Tubes Chamber", CMS NOTE 1998/064
[2] D. Stickland et. al., "CMS Reconstruction Software: The ORCA Project", CMS IN 1999/035
[3] G.Antchev et al. , "A Software Approach for Readout and Data Acquisition in CMS", RealTime 99 Conference, Santa Fe, June 1999
[4] M. Bellato et al., "Java based run control for CMS small daq systems", *Same Conference*