Run Control and Monitor System for the CMS Experiment

L. Berti, M. Biasotto, D. Ceccato, E. Ferro, M. Gulmini, G. Maron, N. Toniolo, S. Ventura, L. Zangrando

INFN, Laboratori Nazionali di Legnaro

I. INTRODUCTION

The Data Acquisition (DAQ) of the CMS Experiment [1] is composed by $O(10000)$ network objects spread in $O(1000)$ nodes, most of them based on commodity PCs, connected to a very efficient Local Area Network. The DAQ complex (both hardware and software) must be controlled in order to synchronize all the objects to perform in the proper sequence the needed actions: configure the experiment, start, stop, etc. The software procedures needed to control the experiment are usually named Run Control System. Moreover both hardware and software needs to be continuously monitored to check their functionality.

This report describes briefly the proposed architecture of Run Control and Monitor System (RCMS) of the CMS experiment and the parts have been prototype.

RCMS operates in a context composed by the following sub-systems of the CMS apparatus:
- Event Builder (EB)
- Trigger (TRG)
- Event Filter (EF)
- Detector Control System (DCS)
- Computing Services (CS)
- User Interface (UI)

All the subsystems are connected together and to the RCMS via a large local switched Ethernet network (DAQ Service Network) that composes the control and monitor network of the apparatus. The same network is also used as data network when some detector partitions are working in standalone mode.

Commands are originated from the UIs to the peripheral subsystems with RCMS acting as a command broker: the macro commands provided by the UIs are interpreted, expanded and routed to the proper sub-system targets by RCMS components. Information status of the action related to the submitted command is in general expected asynchronously (see later in this chapter) and it is handled and logged by RCMS. In the same way RCMS handles all the information provided by the sub-systems and concerning their internal status, malfunctions, errors and, when required, monitor data. RCMS is then a logical central point of control and monitor of a largely distributed system that should feature security capability in order to identify the users of the apparatus and provide them the proper profile to access to the system, log ability to track the information coming from the peripheral sub-systems, logic functionality to manipulate the commands received from the UIs and then to instruct the related actions to the involved target sub-systems. Due to the dimension of the entire system and thus to the number of network objects to control (order of tens of thousands) the RCMS scalability, both in terms of commands distribution capability and of transactions handled per second, is one of the most important design issue.

Seen from this point of view the RCMS information model is very close to a typical transactional model used in the e-business word and it is our intention to exploit the technological developments related to that word both to help the design and the implementation of our system. In fact these technologies are clearly dominating most of the software based business word, which has its basic requirements serving quickly very large databases; connecting many clients, distributed in large geographical area, to those servers; providing, for the nature of the application, safe accesses; scaling easily up in terms both of connectivity to the servers and transactions per second of the data bases; etc. Requirements, as we have just seen, not too far away from the one we have in designing our control and monitor of the experiment. Moreover many of these widely spread software tools follow an open source approach, which makes their integration into our system completely under control given the availability of the sources.

II. RCMS ARCHITECTURE

Figure 1 shows the basic logical layout of the Run Control and Monitor System (RCMS). It consists of 3 types of elements: the session manager, the sub-system controller and a set of services needed to support specific RCMS functionality like security, logging services, etc.

Fig.1 RCMS logical layout
The services that have been identified, and thus defined, so far are:

- **SECURITY SERVICE (SS).** It provides login procedures to RCMS and all the needed functions to manage user accounts. It handles the UserDB data base.

- **RESOURCE SERVICE (RS).** It provides access to the backend configuration data base (ConfigDB) where are stored information concerning the defined sessions, sub-system partitions, DAQ resources description and related hardware configuration. This service handles also the global “still alive heart beat” of the entire CMS DAQ.

- **INFORMATION AND MONITOR SERVICE (IMS).** It collects messages and monitor data coming from any DAQ resources or RCMS internal components and stores them in the logDB data base. IMS messages are classified in types: errors, warnings and generic information. IMS can distribute such messages, with the possibility to filter them according to their type, level of severity, source of the message, etc., to any external subscriber. A similar mechanism is also used to distribute monitor data. IMS handles even the run bookkeeping of the experiment.

- **JOB CONTROL (JC).** It starts, monitors and kills the software infrastructure of RCMS, including the software agents running in the single DAQ resources

- **PROBLEM SOLVER (PS).** It uses the information provided by the RS and by IMS to catch severe malfunctions of the whole apparatus and try to fix them.

### III. RESOURCE SERVICE PROTOTYPE

A Resource Service (RS) has been prototyped to test functionalities and performances. RS handles all the resources (hardware and software) of the DAQ system and then it is basically composed by a Database and its related access APIs. RS handles also the partitions of the system and different concurrent run sessions can be defined.

RS is based on client-server architecture. At the server side all RS functionalities and the access to the back-end DB are implemented by a java-servlet application that runs in a servlet container like Apache Tomcat. The client-server communication protocol called RSP (Resource Service Protocol), uses XML documents as a transfer mechanism for structured data. RSP describes all DAQ resources and it is defined by a XML Schema (W3C Recommendation). All XML documents are stored in a eXist DB (http://exist.sourceforge.net), an Open Source native XML database that implements the XML:DB API for Java programming (http://www.xmldb.org). Alternatively a relational DB can be used.

Any clients (C++, Java, HTML) that handles a HTTP connection can access to the service. Fig.2 depicts the RS prototype.

![Fig. 2. RS Prototype](image)

In order to make the service access easier to the client, a Java API has been defined and implemented. The main purpose of this API is to provide a way to simplify the binding of Java classes to XML document and vice versa. A very useful tool, CASTOR, has been used (http://www.castor.org). CASTOR is an XML databinding framework very easy to use, unlike the two main XML APIs DOM and SAX. The conversion between Java and XML (marshalling and unmarshalling) is fast and can be marshalled any arbitrary Object to and from XML. APIs for C++ and HTML are in progress.

The RS prototype is running and it is being used in real DAQ systems dedicated to test beams and detector validations [2].

---


[2] XDAQ reference