The CMS Data Acquisition System Software

G Bauer[1], U Behrens[2], K Biery[3], J Branson[4], E Cano[5], H Cheung[3], M Ciganek[5], S Cittolin[5], J A Coarasa[4,5], C Deldicque[5], E Dusinberre[4], S Erhan[5,6], F Fortes Rodrigues[7], D Gigi[5], F Glege[5], R Gomez-Reino[5], J Gutleber[5], D Hatton[2], J F Laurens[5], J A Lopez Perez[5], F Meijers[5], E Meschi[5], A Meyer[2,5], R Mommsen[3], R Moser[5,8], V O'Dell[3], A Oh[5], L B Orsini[5], V Patras[5], C Paus[1], A Petrucci[4], M Pieri[4], A Racz[5], H Sakulin[5], M Sani[4], P Schieferdecker[5], C Schwick[5], D Shpakov[3], S Simon[4], K Sumorok[1] and M. Zanetti[5]

[1]MIT, Cambridge, USA; [2]DESY, Hamburg, Germany; [3]FNAL, Chicago, USA; [4]UCSD, San Diego, USA; [5]CERN, Geneva, Switzerland; [6]UCLA, Los Angeles, USA; [7]CEFET/RJ, Brazil; [8]also at Technical University, Vienna, Austria

E-mail: Johannes.Gutleber@cern.ch

Abstract. The CMS data acquisition system is made of two major subsystems: event building and event filter. The presented paper describes the architecture and design of the software that processes the data flow in the currently operating experiment. The central DAQ system relies on industry standard networks and processing equipment. Adopting a single software infrastructure in all subsystems of the experiment imposes, however, a number of different requirements. High efficiency and configuration flexibility are among the most important ones. The XDAQ software infrastructure has matured over an eight years development and testing period and has shown to be able to cope well with the requirements of the CMS experiment.

1. Motivation

There are a number of issues besides high data rates and volumes that make data acquisition systems for high-energy physics unique [1, 2]. A diverse set of application scenarios and an ever-changing environment are just some of the items that need to be mastered. We concluded that a purpose-built application would not scale from the early design and construction stage to a system required for operating the CMS experiment and that an alternative approach was needed for building a distributed data acquisition system running on O(5000) computers with O(15000) application components [3].

Our vision, based on the 40 year old and well-proven theory of mass-produced software components [4-6] was to develop a homogeneous architecture for data acquisition that can be used in various application scenarios, scaling from small laboratory environments to the large, collaboration-based experiment. This goal is desirable for numerous reasons, most importantly, that with developers being highly heterogeneous and high-energy physics being a niche, it would be useful to concentrate expert knowledge in a single place. Developers can profit from this knowledge that is cast into an infrastructure leading to effective implementation and integration. Efficiency enablers built into the software base allow users to profit from good performance in a variety of operating conditions. Software that is used in a larger context is better understood and can reach a higher level of stability. By developing along these lines, the software infrastructure became an integral part of the architectural scaffolding. Design pattern implementations guide developers and reduce tedious and error prone work of designing data acquisition applications. Consequently the transition from initial test and evaluation setups to production systems is simplified.

2. Requirements

We identified core requirements that have to be met by a general data acquisition software environment during the initial phase of the development [1]. They were the basis for the architecture of the CMS online-software, as it exists today.

The functional requirements relate to the tasks of the system, whereas the non-functional ones capture aspects that stem from environmental constraints. The software must provide the means for movement of data, execution and steering of applications and a baseline set of application components to perform reoccurring data acquisition tasks (e.g. event-building). True interoperability calls for decoupling of application code from protocol code at run-time such that communication at the application level can be performed in the same way even if the underlying protocols are changed. Communication over multiple transport protocols and data formats concurrently must be supported. Applications need a set of functions to access custom electronics devices at user level for configuration, control and readout purposes. The infrastructure must include a homogeneous way to describe all hardware and software components that make up the system and how they are interconnected. Such a language is the basis for a configuration tool that can adapt to a variety of different applications and use-cases [7]. All information about the system and its components produced at run-time must be accessible for monitoring and for tracking errors. Thus, services must be present to record different types of information, such as logging messages, error reports, as well as composite data types.

A domain specific framework is a big step towards easing the construction of data acquisition systems by non-experts. Even larger benefits can be obtained by providing generic application components that make use of the described services to perform the following, re-occurring tasks:

- Collection of data from one or multiple data links to be made available to further components in the processing chain through a single and narrow interface.
- Event building (the combination of logically connected, but physically split data fragments) from multiple data sources on a set of parallel working processing units.
- Near real-time application monitoring
- Error processing in a distributed environment

To let operators interact with the system for configuration, control and monitoring purposes, a user interface that is decoupled from the service implementation must be provided. The design should allow seamless remote control from any place in the world and should accommodate a number of different experiment control solutions, ranging from single computer configurations to the final system that spans several thousands of nodes.

In addition to functions, a number of constraints are imposed on the software. They originate from the diverse environment in which the system is embedded.

The software system must be adaptable to different operating systems and hardware platforms. More than merely accommodating a number of Unix flavours, this feature must support accessing data across multiple bus and switching interconnects and the possibility to add new communication and readout devices without the addition or removal of explicit instructions in user applications. Operating system independence can only be maintained if user applications do not rely on system specific functions. Most importantly, memory management tools of the underlying system should not be exposed directly to applications, since their uncontrolled use affects the robustness of the system.

All system components are unambiguously identified within the system for communication, control and tracking purposes using a unique scheme. Adopting an abstract addressing notation does not give any additional functionality but eases the integration of different technologies and leads to long-term stability of the programs.

Scalability [8] can be described in terms of efficiency and capacity as a function of required resources. Efficiency represents the amount of resources needed to deliver a unit of service, e.g. the building of a physics event from data fragments in a networked system. Capacity represents the maximum rate of service that a system can handle. A system is considered to be scalable if capacity continues to grow, even slowly, as more resources are put into the system in order to match increased requirements. Data acquisition is dominated by scalability needs in terms of communication and memory usage. Therefore, the design of the system must have constant overhead for each transmission operation that is small with respect to the underlying communication technologies. In addition memory pools with a buffer-loaning scheme can address memory resource scalability issues.

3. Architecture

Based on the outlined requirements, we established a software product line [9] specifically designed for distributed data acquisition systems. This suite, called XDAQ [10], includes design pattern implementations, platform utilities, a distributed processing environment and generic DAQ application components that can be tailored to a variety of application scenarios (see figure 1).

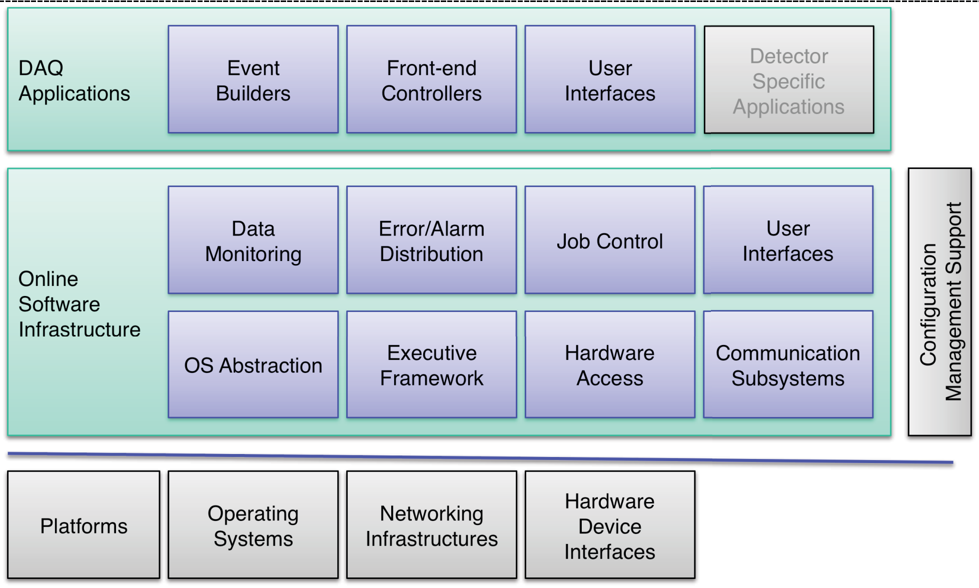


Figure 1. Components of the CMS on-line software infrastructure.

Applying a product line approach to data acquisition aims at shifting the focus from application programming to integration tasks, thus speeding up development and obtaining good performance by using well-established and tested design patterns. The architecture follows a layered component model, splitting the software into three suites: (1) coretools, (2) powerpack and (3) several worksuites.

3.1. Coretools

Coretools builds upon the hardware and operating platform layers. An abstraction layer implements wrappers and adapters for a number of different operating systems. Originally built for VxWorks, Solaris, MacOS X and Linux, the software's maintenance is today limited to the CERN Scientific Linux platform as required by the CMS experiment. A library for accessing VME and PCI devices at user level in an operating system and platform independent manner is a vital component to tailor applications for experiment specific scenarios. Application designers are no longer concerned with operating system specifics, but can concentrate on communicating with their detector hardware. Coretools contains a number of transport technology plug-ins that allow applications to communicate concurrently over different network technologies.

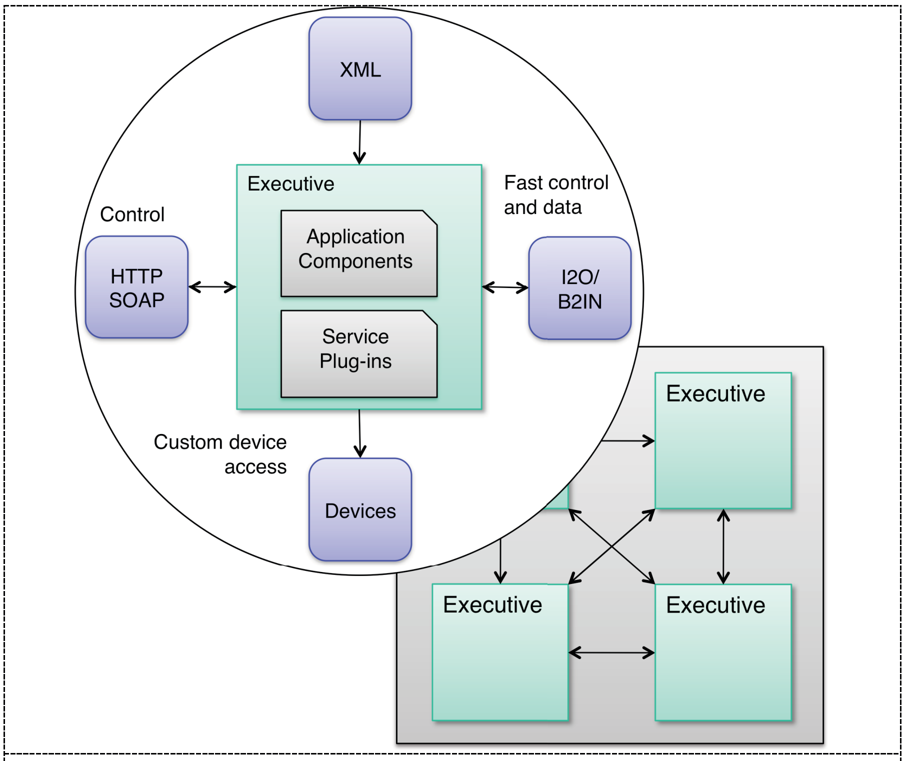


Figure 2. The executive, replicated building block in distributed system

The executive framework is the core of the distributed processing environment for creating networked applications for data acquisition. The core unit of the distributed processing environment XDAQ is the "executive". It provides applications with functions for communication, configuration control and monitoring. Written entirely in C++ with an emphasis on platform independence, it implements well-established techniques and best-practices in embedded and distributed computing to provide applications with efficient, asynchronous communication. It includes memory pools for fast and predictable buffer allocation [11] that can be configured for different kinds of technologies such as virtual, physical or remote/shared memory. It offers support for zero-copy operation [12, 13] and an efficient dispatching mechanism for event-driven processing [14]. At least one copy of the executive process runs on every processing node in the data acquisition system. Applications are modeled according to a software component model [6] and follow a pre-scribed interface. They are compiled and the object code is loaded dynamically, at run-time into a running executive. Multiple application components, even of the same application class may coexist in a single executive process. Pluggable service components called peer-transports can be loaded into the executive to enable the process with the capability to communicate over a specific networking technology. Loading multiple peer-transports allows applications to communicate over different transport technologies concurrently. A common peer-transport implements communication across TCP/IP networks, allowing fine-grained configuration such as the network port specification for individual messages. Peer-transports have been proven valuable during the ten year construction period in which different network technologies were used without having to modify application components. Changing the network is as simple as changing the configuration.
All configuration, control and monitoring is performed through XML documents that can be loaded from a file or received over SOAP and HTTP protocols. A rich set of data structures, including lists and vectors are exportable and can be inspected by clients through the executive SOAP services.

3.2. Powerpack

Powerpack extends coretools with services that are orthogonal to applications, frameworks and libraries. The two major elements of the Powerpack are scalable monitoring and error-processing.

XMAS (XDAQ Monitoring and Alarming System) is based on a service oriented architecture [15], in which a 3-tier structured collection of communicating components cooperate to perform the monitoring task. As shown in Figure 3, the system builds upon a scalable publisher-subscriber service consisting of a pool of eventing applications. A broker application balances the load among the services. DAQ applications act as data producers through sensor services modules to publish monitoring data. Similarly sentinel services are used to report errors and alarms. Other services for processing, storing, filtering and transforming the information express their interest by selectively subscribing to eventing services. Presentation components can either subscribe or directly retrieve monitoring data from the required provider services. All services are relocatable and run independently of each other without a need for external control. Communication among services is established through a rendez-vous mechanism with the help of discovery services facilities [16]. The heartbeat service keeps track of all running services and DAQ applications.

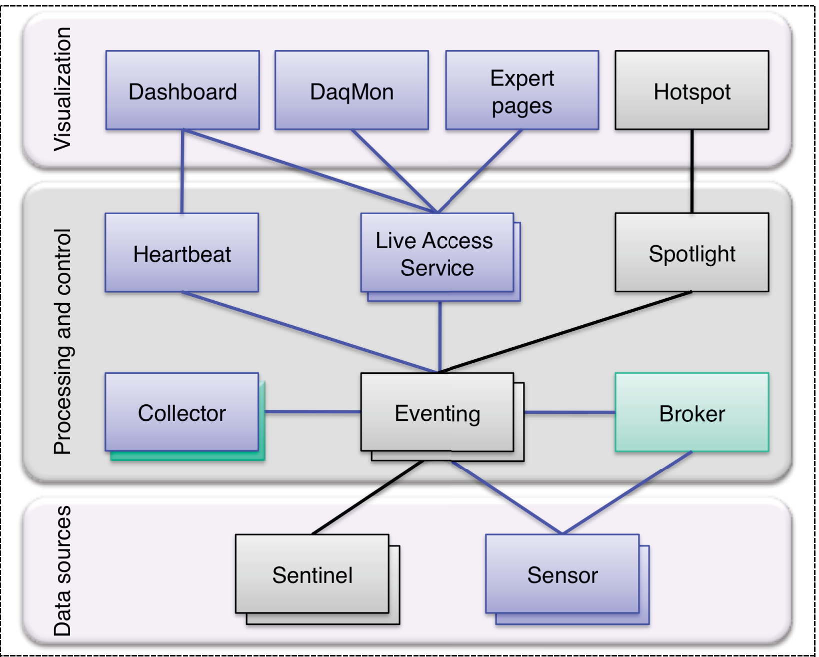


Figure 3. XMAS architecture

All metrics to be merged during the collection process are treated using a uniform, table based data format. Table definitions enumerating data items, called flashlists, are specified in XML. Flashlist specifications reliably identify the content for merging, tracking and analysis with additional information, including timestamps, version and application identification fields. The framework transparently inserts these data. Collected data is served to user interface applications on request in JSON, XML, CSV and SunRPC binary format by the life access services over HTTP protocol.

DAQ applications notify exceptional conditions asynchronously using a data format similar to that of the monitoring infrastructure. Two different scenarios can be identified. Applications that detect persistent deviations from normal behavior can report errors. For a temporary deviation an alarm can be fired and eventually revoked when the asserted condition is resolved. Reporting errors and alarms is performed through sentinel services that take care of reliably routing notifications and preventing flooding the system. Reports are recorded in a database by a service called spotlight. This allows playing back the history of events. Errors and alarms are visualized by the hotspot facility that maps them to graphical elements using Adobe Flex [17] according to user defined system models.

JobControl is a service running on each computer for starting applications and tracking their process descriptor states. In case of an abnormal termination, the service reports an error via the monitoring and alarming system. One JobControl is started per host automatically during the host boot sequence. JobControl applications expose their services through a SOAP interface.

3.3. Worksuite

Worksuites are collections of application components that have been implemented with XDAQ. Although some of the components are not specific to the CMS experiment, all applications are today bundled in a single worksuite. Examples include a generic event builder, controllers for CMS detector readout hardware, a SOAP gateway to the PVSS II and SMI++ statemachine software [18].

4. Achievements

4.1. Generic Event Builder

One of the worksuite applications is a generic event-builder. This distributed application implements a task that is common to many high-energy physics experiments [19, 20]. It assembles event data fragments from detector readout computers and delivers complete physics events. The application neither relies on a specific event data format, nor is it concerned with any experiment specific interfaces. It can be used in a variety of scenarios and experiments by tailoring components at the readout and trigger interface.

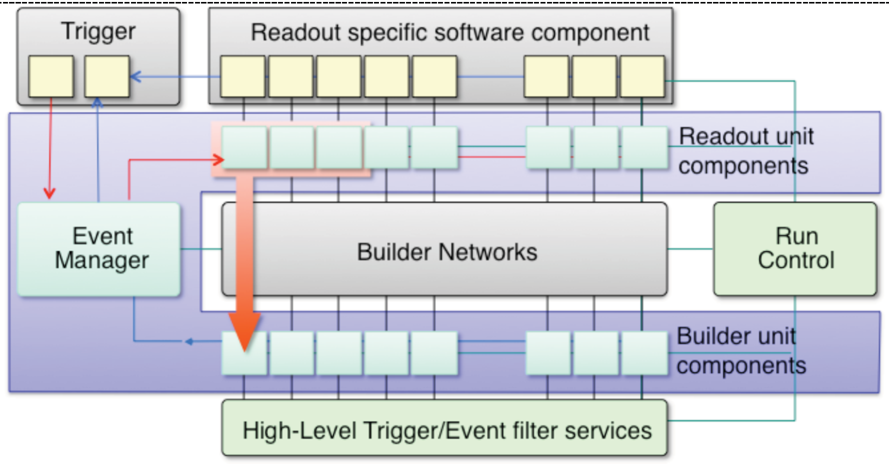


Figure 4. Event-builder application

The event builder consists of three collaborating components (see figure 4): a readout unit (RU), a builder unit (BU) and an event manager (EVM). Data that are emitted by customized readout devices are forwarded to the readout unit application that is replicated over a number of computers, depending on the performance needs of the application scenario. A RU buffers data from subsequent single physics events until it receives a control message to forward a specific event fragment to a builder unit. One builder unit collects event fragments belonging to a single collision event from all RUs and combines them into a complete event. BU applications are replicated over a number of computers to fit the throughput requirements. The BU exposes an interface to event data processors, called filter units (FU). This interface can be used to make event data persistent or to apply event-filtering algorithms. Such, event-level parallelism can be exploited in computing farms by independently processing events on a large number of computers. The EVM interfaces trigger readout electronics and

controls the event building process by mediating control messages between RUs and BUs. The event-building protocol, interfaces and configuration parameters are open source and well documented through a document on the CERN EDMS system [21].
Scalability and adaptability are paramount to event-building [22, 23]. Leveraging the capabilities of the XDAQ framework, the event-builder application can scale in two dimensions (see figure 5). Firstly the number of readout units and builder units can be changed to fit the processing and memory requirements of the setup, ranging from small laboratory setups to the final experiment that comprises thousands of application components. Secondly, depending on data throughput needs, communication paths can be configured through the pluggable peer transports. To increase throughput, multiple Gigabit Ethernet ports can be used to inject data from the RUs into the switching fabric. A larger number of BUs picks them up using fewer legs per host (trapezoidal configuration).

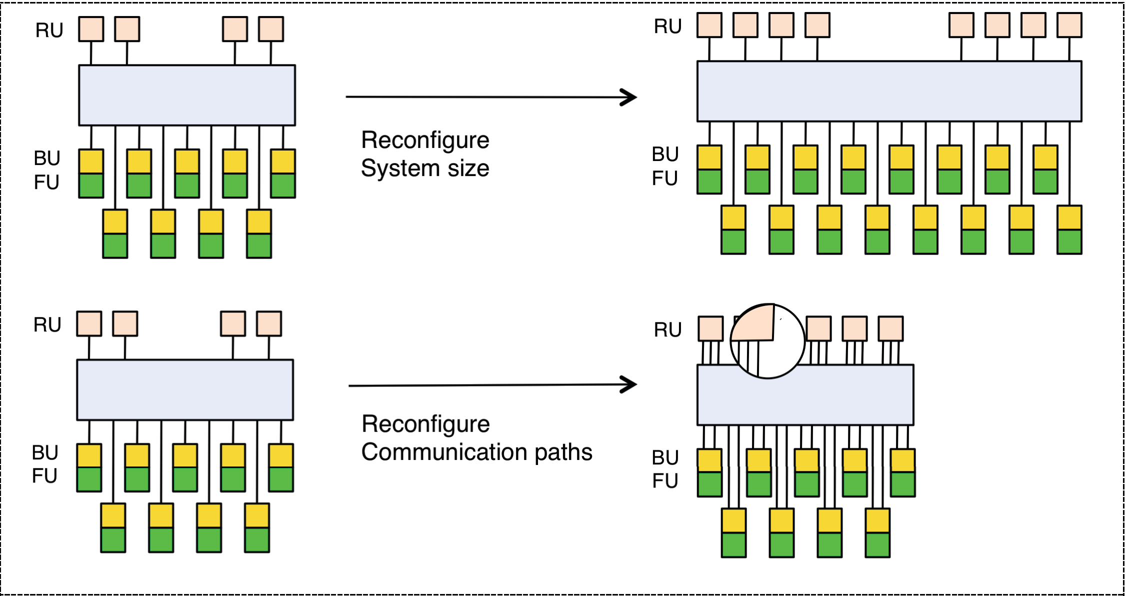
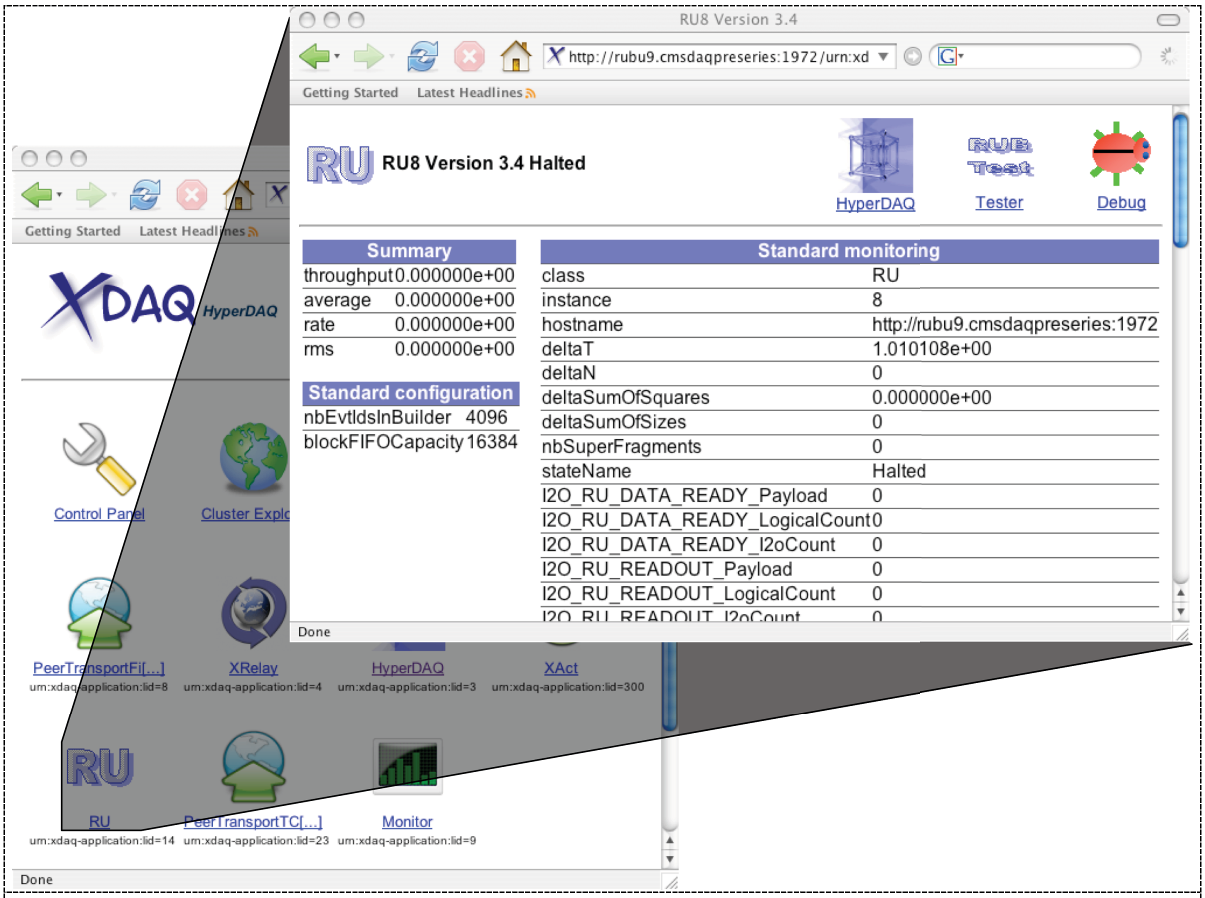


Figure 5. Scaling up the event-builder

4.2. HyperDAQ

HyperDAQ marries two well-established technologies to provide easy access to distributed computing systems: the World Wide Web and Peer-to-Peer systems [24]. An embedded HTTP protocol engine turns an executable program into a browsable Web application that can serve application specific data to clients in different formats (e.g. HTML, XML, plain text, JSON, SunRPC binary format). This is achieved through a data serialization engine that allows adding data formats through a plug-in interface. While Web pages contain hyperlinks that have been inserted at the time of page creation, HyperDAQ can also present links to data content providers when they become available. Applications and data resources are uniquely identified through a uniform scheme based on URI and URN formats. There is another advantage of this concept, which represents a new way of interacting with a distributed system: Traditionally systems give the user a single point of access. With HyperDAQ, any node in the cluster may become an access point from which the whole distributed system can be explored. Presenting links to other applications permits navigating through the distributed system from one application to another. With the additional SOAP communication facilities and the capability to quickly include application specific callback handlers in application components, this technology goes beyond controlling and inspecting single executives by allowing a seamless step up to a Web based experiment control system using widely available tools.


RU8 Version 3.4
http://rubu9.cmsdaqpreseries:1972/urn:xd
Getting Started
Latest Headlines
XDAQ HyperDAQ
Control Panel
Cluster Explo
PeerTransportFi[...]
urn:xdaq-application:lid=8
XRelay
urn:xdaq-application:lid=4
HyperDAQ
urn:xdaq-application:lid=3
XAct
urn:xdaq-application:lid=300
RU
urn:xdaq-application:lid=14
PeerTransportTC[...]
urn:xdaq-application:lid=23
Monitor
urn:xdaq-application:lid=9
HyperDAQ
Tester
Debug
Done

RU8 Version 3.4 Halted

Summary	
throughput	0.000000e+00
average	0.000000e+00
rate	0.000000e+00
rms	0.000000e+00

Standard configuration	
nbEvtIdsInBuilder	4096
blockFIFOCapacity	16384

Standard monitoring	
class	RU
instance	8
hostname	http://rubu9.cmsdaqpreseries:1972
deltaT	1.010108e+00
deltaN	0
deltaSumOfSquares	0.000000e+00
deltaSumOfSizes	0
nbSuperFragments	0
stateName	Halted
I2O_RU_DATA_READY_Payload	0
I2O_RU_DATA_READY_LogicalCount	0
I2O_RU_DATA_READY_I2oCount	0
I2O_RU_READOUT_Payload	0
I2O_RU_READOUT_LogicalCount	0
I2O_RU_READOUT_I2oCount	0

Figure 3. HyperDAQ interface of the executive and a readout-unit application

5. Development facts

The CMS on-line software is entirely implemented in C++. Table 1 gives an overview of the currently active code, excluding external libraries. The software is delivered in binary and source code formats for the CERN Scientific Linux platform using the YUM and Quattor software distribution tools. The software covers about one hundred RPM packages including external libraries. The on-line software includes run-time compatibility checking so that independent upgrade and rollback of single packages can be performed at any time. This feature has been shown to be highly valuable for resolving isolated problems with minimal impact on the remaining software.

Table 1. Lines of C/C++ code and number of packages as of February 2009

Suite	.cc	.h	**Sum**	Packages	**RPMs**
Coretools	62’304	43’169	**105’473**	9	**20**
Powerpack	57’031	18’786	**75’817**	5	**34**
Worksuite	138’166	76’875	**215’041**	34	**43**
Total	257’501	136’636	**396’331**	48	**97**

Since the very early days of development, XDAQ has been an open source project at Sourceforge.net. While releases are made available infrequently on Sourceforge, feature and bug tracking have been in use without interruption since the year 2000. The software is developed according to a configuration management plan that takes this environment into consideration [25]. XDAQ development started two years after market surveys and technology studies led to the conclusion that at that time no single tool was able to meet the requirements outlined in this paper. Initially the focus was on performance and flexibility. With the results published in the technical design report in 2002 [3] matching the needs of the experiment, XDAQ was adopted as the experiment-wide recommended infrastructure for developing on-line software in 2003. At that time, Web technologies have been fully integrated and the focus shifted on providing production quality software and applications targeting stability and features that go beyond the quest for efficiency. The software has been used successfully in commissioning the CMS experiment in 2007, culminating in the two week data-taking period after the first beam event in September 2008.

About eight full-time employees developed the code over a period of eight years with an average additional effort of two visiting full-time programmers working with the development team. The tasks of the personnel included design, planning, development, documentation, daily support and maintenance as well as consultancy to other users of the software, namely the CMS subdetector groups. Staff personnel delivered the production quality code, while visiting personnel mainly contributed to testing, prototyping, and evaluation and feasibility studies.

6. Summary

The CMS data acquisition system and subdetector on-line applications have been implemented using a software infrastructure called XDAQ. This software product line is the result of an eight year development process and has proven its fitness for operation with the acquisition of the first beam events on September 10, 2008. This paper summarized key requirements and outlined the resulting architecture of the CMS on-line software infrastructure. Its functional aspects cover:

- protocol and technology independent communication
- user-level transparent access to custom devices
- uniform schemes for configuration, control and monitoring
- provision of generic application components for re-occurring tasks such as event-building
- accessibility through Web technologies (HyperDAQ)

Non-functional aspects that are covered by the online software infrastructure include

- adaptability to multiple platforms
- invariance of application code with respect to underlying implementations
- scalability enablers, such as low latency communication and buffer-loaning memory pools
- flexibility to use multiple different communication channels concurrently and
- identification of all components in the system through a unique addressing scheme.

Acknowledgements

This work was supported in part by DOE and NSF (USA) and the Marie Curie Program.

References

[1] Gutleber J, Murray S, Orsini L 2003 Towards a homogeneous architecture for high-energy physics data acquisition systems *Elsevier Comp. Phys. Comm.* **153(2)** 155-163

[2] Gutleber J, Moser R, Orsini L 2007 *Data Acquisition in High Energy Physics* Proc. Astronomical Data Analysis Software and Systems (ADASS 23-26 Sept 2007) XVII, ASP vol. 394 pp 47

[3] CERN 2002 *Data Acquisition & High-Level Trigger, Technical Design Report* CMS TDR 6.2, LHCC 2002-26 (ISBN 92-9083-111-4)

[4] McIllroy M D 1968 *Mass-produced software components* in: Software Engineering Concepts and Techniques, eds. Buxton J M, Nauran P and Randell B, Reprinted proceedings of the 1968 and 1969 NATO Conferences, Petrocelli/Charter (ACM Press, 1076) pp 88-98.

[5] Parnas D L 1979 Designing Software for Ease of Extension and Contraction, *IEEE Trans. Softw. Eng* **SE-5(2)** 128-137

[6] Nierstrasz O, Gibbs S and Tsichritzis D 1992 *Component-oriented software development* Comm. ACM **35(9)** 160-164
[7] Bauer G et al 2009 Dynamic configuration of the CMS Data Acquisition cluster *J. Phys. Conf. Ser.* (same conference, in print)
[8] Grama A Y, Gupta A and Kumar V 1993 *Isoefficiency: measuring the scalability of parallel algorithms and architectures*, IEEE Par. & Distr. Tech.: Systems & Applications **1(3)** 12-21
[9] Clements P, Northrop L 2002 *Software Product Lines* Addison-Wesley (ISBN 0-201-70332-7)
[10] Gutleber J and Orsini L 2002 *Software Architecture for Processing Clusters based on I_2O* Clust. Comp., J. of Netw., *Software and Applications* Kluwer Acad. Pub., **5(1)** 55-65
[11] Fujimoto R M and Panesar K S 1995 *Buffer management in shared-Memory Time Warp Systems* ACM SIGSIM Sim. Digest, **25(1)** 149-156
[12] Wind River Systems Inc. 1999 *Network Protocol Toolkit, User's Guide V 5.4* ed. 1 Part # DOC-12820-ZD-03 500 Wind River Way, Alameda, CA 94501-1153, USA
[13] Thadani M and Khalidi 1995 *An efficient zero-copy I/O framework for UNIX* Tech. Rep. SMLI TR95 -39 Sun Microsystems Lab Inc. USA
[14] Bershad B N et al 1995 Extensibility, Safety and Performance in the SPIN Operating System *in Proc. 15th ACM SOSP* pp 267-284
[15] Booth D. et al. 2004 Web Service Architecture http://www.w3.org/TR/ws-arch
[16] Guttman E, Perkins C, Vaizades J and Day M 1999 *Sevice Location Protocol Version 2* Internet RFC http://www.ietf.org/rfc/rfc2608.txt
[17] Adobe Flex 3 *Rich Internet Applications* http://www.adobe.com/products/flex/
[18] Franek B and Gaspar C 1998 *SMI++ object oriented framework for designing and implementing distributed control systems* IEEE Trans. Nucl. Sci. **45(4)** 1946-1950
[19] Barsotti E, Booth A and Bowden M 1990 *Effects of Various Event Building Techniques of Data Acquisition System Architectures* Fermilab note FERMILAB-CONF-90/61, Batavia IL, USA
[20] Antchev G et al 2001 The Data Acquisition System for the CMS Experiment at LHC *in Proc. 7th Intl. Conf. Adv. Tech. and Particle Phys. Villa Olmo, Como, Italy (Oct. 15-19, 2001)* World Scientific Publishers (ISBN 981-238-180-5)
[21] Murray S 2007 *RU Builder User Manual* CERN EDMS ID 875261, http://edms.cern.ch/document/875261/1.6
[22] Antchev G et al 2001 *The CMS Event Builder Demonstrator and Results with Myrinet*, Comp. Phys. Comm. **140(1-2)** 130-138
[23] Bauer G et al 2008 *CMS DAQ Event Builder Based on Gigabit Ethernet* IEEE Trans. Nucl. Sci. **55(1)** 198-202
[24] Gutleber J et al 2005 HyperDAQ Where Data Acquisition Meets the Web *Proc. 10th Intl. Conf. Accel. and L. Exp. Phys. Control Sys.* (Geneva, Switzerland, 10-14 October 2005)
[25] Gutleber J and Orsini L 2007 *TriDAS Configuration Management Plan*, CERN EDMS ID 836826, http://edms.cern.ch/document/836826/1.3