HYPERDAQ – WHERE DATA ACQUISITION MEETS THE WEB

R. Arcidiacono[8], V. Brigljevic[9], E. Cano[5], S. Cittolin[5], S. Erhan[6], D. Gigi[5], F. Glege[5], R.Gomez-Reino[5], M. Gulmini[3,5], J. Gutleber[*5], C. Jacobs[5], P. Kreuzer[1], G. Lo Presti[5], N. Marinelli[2], G. Maron[3], F. Meijers[5], E. Meschi[5], S. Murray[5], A. Oh[5], L. Orsini[5], M. Pieri[7], L. Pollet[5], A. Racz[5], P. Rosinsky[5], C. Schwick[5], P. Sphicas[1,5], J. Varela[4,5]

[1]University of Athens, Athens, Greece; [2]Institute of Accelerating Systems and Applications, Athens, Greece; [3]INFN - Laboratori Nazionali di Legnaro, Legnaro, Italy; [4]LIP, Lisbon, Portugal; [5]CERN, Geneva, Switzerland; [6]University of California, Los Angeles, Los Angeles, California, USA; [7]University of California, San Diego, Sandiego, California, USA; [8]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA; [9]Rudjer Boskovic Institute, Zagreb, Croatia

ABSTRACT

HyperDAQ was conceived to give users access to distributed data acquisition systems easily. To achieve that, we marry two well-established technologies: the World Wide Web and Peer-to-Peer systems. An embedded HTTP protocol engine turns an executable program into a browsable Web application that can reflect its internal data structures using a data serialization package. While the Web is based on static hyperlinks to destinations known at the time of page creation, HyperDAQ creates links to data content providers dynamically. Peer-to-Peer technology enables adaptive navigation from one application to another depending on the lifetime of application modules. Traditionally, distributed systems give the user a single point of access. We take a radically different approach. Every node may become an access point from which the whole system can be explored.

INTRODUCTION

Much has been written about the potential benefits of distributed computing systems for high-performance applications including LHC scale data acquisition (DAQ) [1-6]. However, this potential will not be realized if users cannot access online information as easy as a click of the mouse. To reach this goal, we took a fresh look at an already well-understood technology, the World Wide Web [7].

A data acquisition system for an LHC scale high-energy physics experiment can be architected as a distributed computing system (see figure 1). In a DAQ computing cluster, a set of application modules, distributed over the computers in the system, collaborate to perform the data acquisition task. In contrast to number-crunching systems that dominate the parallel-computing domain, a distributed system for data acquisition is dominated by throughput and scaling requirements [1, 8]. A distributed event builder is an example for this. *Readout units* acquire and buffer a piece of the whole experiment's output each time a particle collision is accepted by the trigger processor. All pieces of an event are assembled in a *builder unit* that serves the data for further processing with higher-level trigger algorithms to another distributed system, the *event filter farm*. The DAQ system is embedded in the experiment and thus interacts with custom hardware, the detector readout and the trigger system. Tuning the system is key to assure uninterrupted data taking at good performance levels and to consequently successfully operate the experiment. Hence the requirement for interaction capabilities that adapt to the diverse elements of the system and its size easily.

We use the term *HyperDAQ* to refer to a newly conceived way of giving users access to distributed data acquisition applications. This paper explains the concept behind HyperDAQ. It presents its realization in XDAQ [3], a distributed data acquisition toolkit developed at CERN and sheds light on related topics. They cover access control and security, live data distribution for monitoring purposes and Web based application development. Finally we outline a use-case implemented for the Compact Muon Solenoid experiment [9] at CERN.

* Corresponding author: Johannes.Gutleber@cern.ch

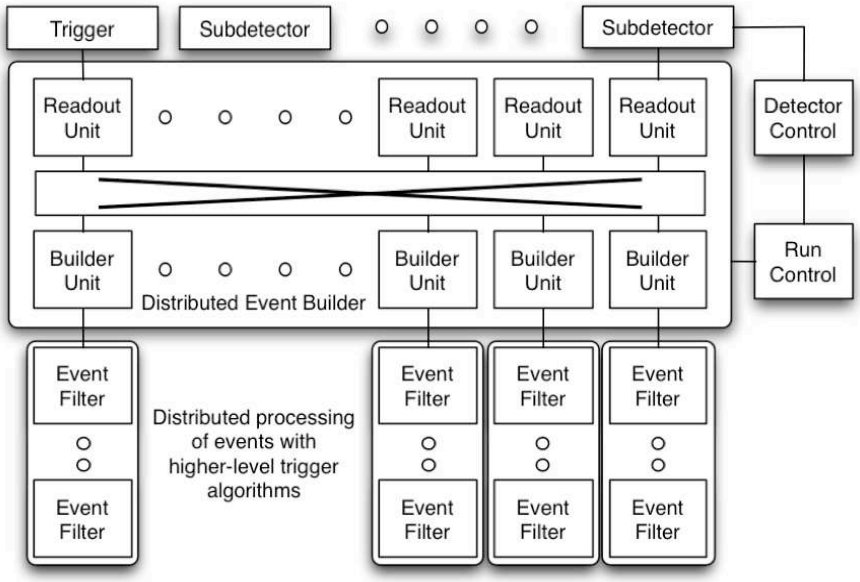

Figure 1: A generic distributed data acquisition system for a high energy physics experiment.

CONCEPTS

HyperDAQ marries two well-established technologies to provide access to a distributed computing system: the World Wide Web [7] and Peer-to-Peer systems [10]. It was conceived to give users access to distributed DAQ applications easily. HyperDAQ has been implemented as an extension to XDAQ, a C++ toolkit for clustered data acquisition developed by us at CERN [11]. An embedded HTTP protocol engine turns an executable program into a browsable Web application that can serve application specific data structures to clients in different formats. This is achieved through a data serialization engine that allows adding data serialization formats through a plug-in interface. While Web pages contain static hyperlinks to other pages that have been inserted at the time of page creation, HyperDAQ presents also links to data content providers when they become available. Peer-to-Peer technology [12] is used to discover data content providers such as other DAQ applications, thus alleviating users from hand-coding hyperlinks and from knowing the exact providers and their physical locations before operating the system. There is another advantage of this concept, which represents a radical 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 browsable applications permits navigating through the distributed system from one application to another.

XDAQ

XDAQ is a programming toolkit designed specifically for the development of distributed data acquisition systems. Its main goals are to provide a homogeneous architecture for developing application components that interact with each other to perform highly efficient data acquisition tasks and to abstract from details of the underlying hardware. Meeting both requirements at the same time has long been thought to be contradictory. Evidence has been provided that the toolkit indeed fulfils the requirements [13].

XDAQ includes an *executive* process that runs on every node in the data acquisition network. Applications are compiled and the object code is loaded dynamically into a running executive. Multiple application components, also of the same application type may coexist in a single executive process. The executive provides applications with communication, configuration and control services [1-3]. Written entirely in C++ with an emphasis on platform independence, it implements well-established techniques to provide applications with efficient, asynchronous communication. They include memory pools for fast and predictable buffer allocation [14], support for zero-copy operation [15,16] and an efficient dispatching mechanism for an event-driven processing scheme [17]. Applications are modelled according to a software component model [18] and follow a defined-

scheme for describing interfaces. Communication with and among applications takes place through pluggable peer-transport components that provide transport services over some communication protocol, e.g. TCP or Myrinet/GM. Configuration, control and monitoring are usually performed over the SOAP/HTTP [19, 20] protocol. Multiple transports can be used concurrently. A rich set of data structures, including lists, vectors are exportable and can be inspected by clients through data serialization services in various formats. Apart from an XDR binary format [21,22], data structures can also be accessed via SOAP or pure HTTP. Data formats can also be mixed, so that a complex structure can be transmitted in binary as a response to an HTTP request issued by a Web browser.

HyperDAQ

HyperDAQ is the name we gave to a set of technologies that we use to give users access to distributed data acquisition systems:

- an embedded HTTP protocol engine
- URN and URL based identification of applications and application resources
- a dispatcher for incoming requests
- an API to implement dynamically created Web pages
- provision of data serialization methods for binary and XML formats
- a Peer-to-Peer discovery mechanism
- a security mechanism for access regulation based on Web protocols

Through these mechanisms any XDAQ application component becomes accessible through the Web protocol and any XDAQ process may serve as an access point for all other XDAQ applications in the distributed system.

The embedded HTTP engine

The embedded HTTP engine is implemented as a pluggable peer-transport for the XDAQ system. It is loaded through a configuration file that is present on each host when the XDAQ process is launched. It is aimed to be highly efficient, both in terms of processing speed and memory consumption. The engine is built to work together with a dynamically loadable Web security component that is also implemented as an XDAQ application. If the security components presence is detected, the HTTP peer-transport will invoke the security application's configured access policy each time an HTTP request is received. Policies are selectable and range from access regulation based on IP addresses over basic Web authentication to more sophisticated session handling implementations [23]. Additional policies can be added transparently without interfering with the remaining system. Although the embedded HTTP engine is capable to serve files, this capability is merely used to supply style sheets and images to the user that interacts with the system through a Web browser. The main task is to route requests to a dispatcher that invokes user supplied callback functions.

The dispatcher

A dispatcher is invoked when the HTTP engine recognizes a request for information from an XDAQ application resource. A resource is indicated by a Uniform Resource Name (URN) [24]. An URN has three parts: *urn:<namespace identifier>:<namespace specific string>*. The first part is invariant and consists of the text string "`urn`". The second part denotes the type of resource requested, the most important for us being a callback function implemented in an XDAQ application. In this case, the namespace is "`xdaq-application`". The third part contains an implementation specific identifier for the target application component. Its format depends on the specified namespace string.
The dispatcher extracts the name of the callback function that is given after the URN and invokes the function. Runtime exceptions occurring during the processing of the callback that are not caught by the application programmer are intercepted and reported to the user in a tin-canned failure page. Here is an example URL that we can enter in a standard Web browser to retrieve some performance data through HyperDAQ functionality:

```
http://hostname:port/urn:xdaq-application:service=monitor/retrieve?cpuUsage
```

The C++ callback function, "`void retrieve(xgi::Input* in, xgi::Output* out)`" indicated in the example above, has a specified interface that provides the programmer with an input

stream containing all HTTP headers and submitted form data as well as an output stream into which the response data can be written. The format of these data may be anything ranging from HTML code to binary data, depending on the indication that the client gives in his request.

The XGI library

The XGI library comprises a set of functions that aid the user to implement callback functions. It includes header and footer generators, content encoders for serving images and other binary data in MIME formats, menu creation classes, even a complete dynamic content presentation engine.

Data serialization

Structured application data are served through a package called *xdata*. Being part of XDAQ, it is a generic data serialization/deserialization package that can render C++ data types, simple or complex through pluggable serializers. A binary serializer for the XDR rules, also used for remote procedure calls [22] and an XML serializer following the SOAP data type specification [25] are included by default. Further serializers can be written by the user.

One of the cornerstones of this package are *Infospaces*. These are objects that can be shared among application components within the same process. Data written into these spaces can be accessed by other applications. Only this allows creating non-interfering monitoring applications that are decoupled from the specific data taking code. A monitor package that implements this behaviour is included for collecting and presenting data without the need for additional programming.

Peer-To-Peer Discovery

While all previously outlined technologies make any C++ application accessible through a Web browser, it is the use of peer-to-peer discovery mechanisms that truly ease the interaction with the distributed system as a single entity. Each node in the system may discover other resources, such as XDAQ application components. Access to this service is given through a versatile abstract interface, such that the underlying implementation can be exchanged transparently. The current software includes support for the Service Location Protocol (SLP) [26] and Universal Plug and Play (UPnP) [27]. The discovery service implementation includes the concept of peer-groups so that only resources of joined groups are seen. This improves scalability in ever growing distributed systems.

Once resources are discovered, hyperlinks are created that can be provided to users in Web pages. Browsing a single XDAQ process opens thus multiple doors, each one leading to another application resource, irrespective of their physical location.

DAQ MONITORING – USE CASE

A use case where HyperDAQ technology has been proven truly useful is monitoring the Compact Muon Solenoid experiment's data acquisition system at CERN. While the experiment is still under construction it is important to experience the behaviour of the data acquisition components under a variety of configurations and load situations. A generic monitor application runs aside the data taking applications in each XDAQ process in the distributed system. Any of the monitor applications can be elected to act as a data collector. Through the data collector Web interface, the overall status and performance of the system are presented. Dynamically created links to the data taking application components allow the operator to obtain detailed information about specific information for debugging, problem analysis or performance evaluation. The capability of HyperDAQ to serve data not only in HTML format, but also in binary allows users to retrieve monitoring data not only through the Web browser, but also through other means. In one case a Microsoft Excel application retrieves summary data into spreadsheets. Such, daily reports can be generated making use of the spreadsheets' calculation and chart functions. A second user-interface has been created using National Instrument's LabView. With this front-end panel, the instantaneous performance of the event-building application is presented in the provisory control room. Using the content presentation component of XDAQ, real-time video of the machine room is streamed to the control room, together with near real-time data throughput, CPU and memory usage plots (see figure 2). Monitoring data are updated at an interval of 1 second. CPU usage of the data collector is 4% on a dual Xenon, 2.4 GHz machine running Scientific Linux CERN 3, kernel 2.4.21. The memory footprint of the running application is 14.9 Mbytes.

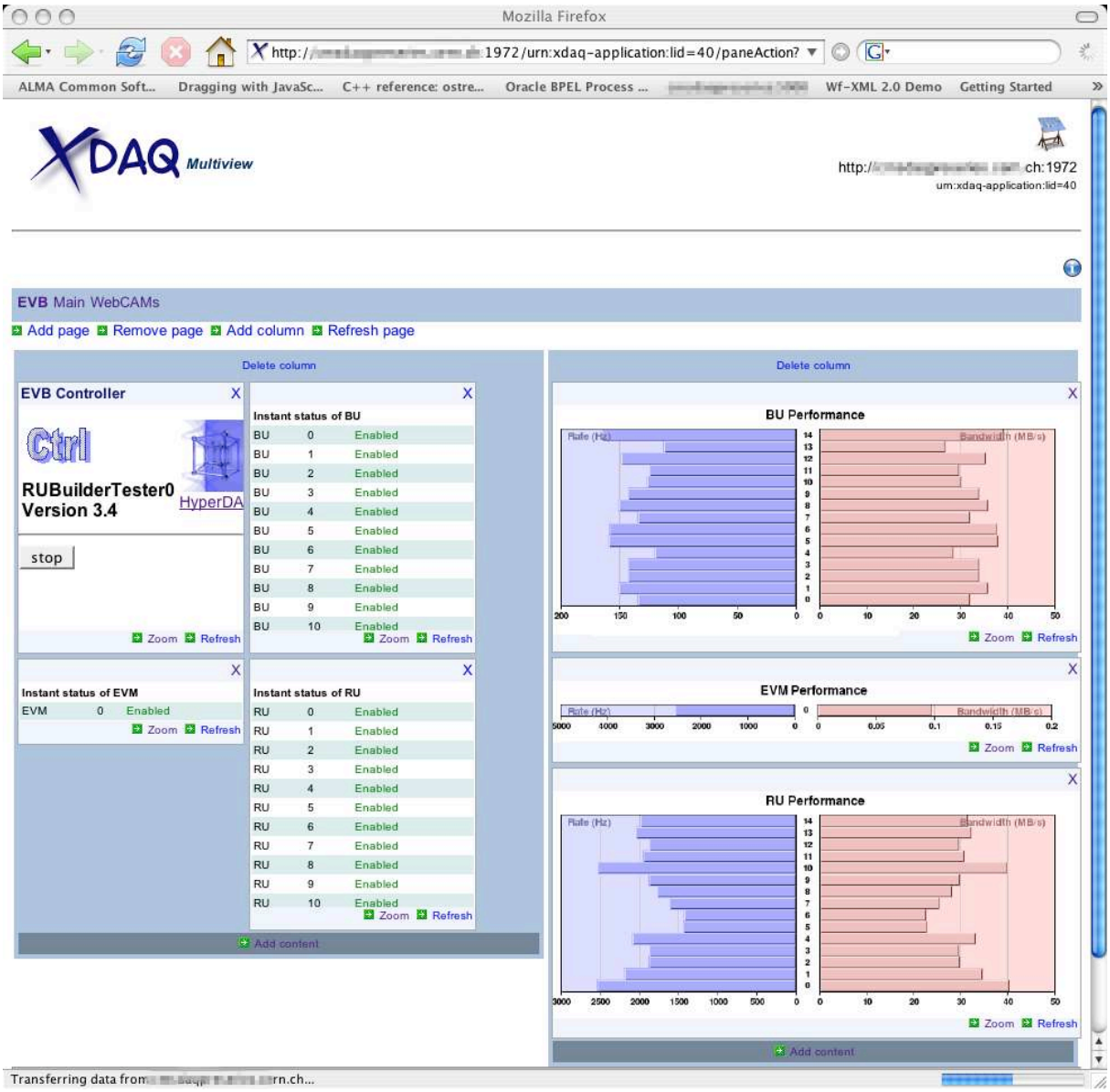

Figure 2: Monitoring data from a cluster of 33 computers presented through HyperDAQ in a browser.

SUMMARY

In this paper, we have argued in general about the necessity of a direct interaction approach with application components in a distributed data acquisition system and presented a Web and Peer-to-Peer based solution – *HyperDAQ*. The implementation of the concept is currently in use for the Compact Muon Solenoid experiment under construction at CERN. We have tried to paint the big picture and identify the key issues. In our group, we are currently setting up the monitoring system for the data acquisition. Subdetector projects start to adopt the technology in their local systems. We believe that a Web based approach facilitates final integration of all data acquisition and detector application components to give the users a single, homogeneous way to interact with the system.

REFERENCES

[1] J. Gutleber, S. Murray and L. Orsini, "Towards a homogeneous architecture for high-energy physics data acquisition systems", Elsevier, Comp. Phys. Comm. 153:155-163, 2003.
[2] J. Gutleber and L. Orsini, "Software Architecture for Processing Clusters based on I2O", Cluster Computing, the Journal of Networks, Software and Applications, Kluwer Academic Publishers, 5(1):55-65, 2002.
[3] G. Antchev et al., Clustered Data Acquisition for the CMS experiment, Computing in High Energy and Nuclear Physics, Beijing, China, Science press, pp. 601-605, September 3-7 2001.
[4] S. Shasharina, R. Eger, J. Cary, "Data grid for fusion simulations and experiments", Elsevier, Comp. Phys. Comm. 164:134-137, 2004.
[5] J.P. Almeida, M.v. Sinderen, D.A.C. Quartel, L.F. Pires, "Designing Interaction Systems for Distributed Applications", IEEE Distr. Syst., 6(3), 2005.
[6] F. Kordon, L. Pautet, "Toward Next-Generation Middleware?", IEEE Distr. Syst., 6(3), 2005.
[7] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk-Nielsen, A. Secret, "The World-Wide Web", Comm. ACM, 37(8):76-82, 1994.
[8] J. Gutleber. "Challenges in Data Acquisition at the Beginning of the New Millennium", Proc. of the 1st Int. Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems, IEEE, Phoenix, Arizona, November 30, 1999.
[9] "Data Acquisition & High-Level Trigger", Technical Design Report, CERN, CMS TDR 6.2, LHCC 2002-26, ISBN 92-9083-111-4, December 2002.
[10] G. Lo Presti, "Peer-To-Peer Architectures in Distributed Data Management Systems for Large Hadron Collider Experiments", doctoral thesis, Universita degli Studi di Palermo, Italy, December 2004 (see also http://cern.ch/lopresti)
[11] See also http://xdaqwiki.cern.ch and http://cern.ch/xdaq
[12] R. Kurmanowytsch. "An Overview of Peer-to-Peer Topologies". Technical Report TUV-1841-2003-04. Distributed Systems Group, Technical University of Vienna. 2003.
[13] G. Antchev et al., "The CMS event builder demonstrator and results with Myrinet", Comp. Phys. Comm., 140(1-2):130-138, 2001.
[14] R.M. Fujimoto and K.S. Panesar, "Buffer management in shared-Memory Time Warp Systems", ACM SIGSIM Simulation Digest, 25(1):149-156, 1995.
[15] "Network Protocol Toolkit, User's Guide V 5.4", edition 1, Part # DOC-12820-ZD-03, Wind River systems, Inc., 500 Wind River Way, Alameda, CA 94501-1153, USA, July 7, 1999.
[16] M. Thadani and Khalidi, "An efficient zero-copy I/O framework for UNIX", Technical Report, SMLI TR95 -39, Sun Microsystems Lab, Inc., May 1995.
[17] B.N. Bershad et al., "Extensibility, Safety and Performance in the SPIN Operating System", in Proc. of the 15th ACM Symposium on Operating system Principles, pp. 267-284, 1995.
[18] O. Nierstrasz, S. Gibbs and D. Tsichritzis, "Component-Oriented Software Development", Comm. of the ACM 35(9):160-164, 1992.
[19] D. Box et al., "Simple Object Access Protocol (SOAP) 1.1", W3C Note 08, May 2000 (see also http://www.w3.org/TR/SOAP).
[20] G. Glass, "Web Services: Building Blocks for Distributed Systems", Prentice Hall, 2002.
[21] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network Filesystem", Proc. USENIX Conf., Portland OR (USA), 119-130, 1985.
[22] R. Srinivasan, "XDR: External Data Representation Standard", Internet RFC 1832, August 1995.
[23] J. Franks et al., "HTTP Authentication: Basic and Digest Access Authentication", Internet RFC 2617, June 1999.
[24] R. Moats, "URN Syntax", Internet RFC 2141, May 1997.
[25] P. V. Piron, A. Malhotra, "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001.
[26] E. Guttmann, et al., "Service Location Protocol, Version 2", Internet RFC 2608, June 1999.
[27] C. Bettstetter, C. Renner, "A comparison of service discovery protcols and implementation of the service location protocol", in Proc. EUNICE Open European Summer School, Twente, Netherlands, Sept. 13-15, 2000.