Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Nuclear Instruments and Methods in Physics Research A 546 (2005) 324–329
NUCLEAR INSTRUMENTS & METHODS IN PHYSICS RESEARCH Section A
www.elsevier.com/locate/nima

Feasibility study of a XML-based software environment to manage data acquisition hardware devices

R. Arcidiacono[a], V. Brigljevic[b,c], G. Bruno[b], E. Cano[b], S. Cittolin[b], S. Erhan[d], D. Gigi[b], F. Glege[b], R. Gomez-Reino[b], M. Gulmini[e,b], J. Gutleber[b], C. Jacobs[b], P. Kreuzer[f], G. Lo Presti[b], I. Magrans[b,g,*], N. Marinelli[h], G. Maron[e], F. Meijers[b], E. Meschi[b], S. Murray[b], M. Nafría[g], A. Oh[b], L. Orsini[b], M. Pieri[i], L. Pollet[b], A. Racz[b], P. Rosinsky[b], C. Schwick[b], P. Sphicas[f,b], J. Varela[j,b]

[a]*Massachusetts Institute of Technology, Cambridge, MA, USA*
[b]*CERN, Geneva, Switzerland*
[c]*Rudjer Boskovic Institute, Zagreb, Croatia*
[d]*University of California, Los Angeles, Los Angeles, CA, USA*
[e]*INFN—Laboratori Nazionali di Legnaro, Legnaro, Italy*
[f]*University of Athens, Athens, Greece*
[g]*Electronic Engineering Department, Universidad Autónoma de Barcelona, Barcelona, Spain*
[h]*Institute of Accelerating Systems and Applications, Athens, Greece*
[i]*University of California, San Diago, San Diago, CA, USA*
[j]*LIP, Lisbon, Portugal*

Available online 7 April 2005

Abstract

A Software environment to describe configuration, control and test systems for data acquisition hardware devices is presented. The design follows a model that enforces a comprehensive use of an extensible markup language (XML) syntax to describe both the code and associated data. A feasibility study of this software, carried out for the CMS experiment at CERN, is also presented. This is based on a number of standalone applications for different hardware modules, and the design of a hardware management system to remotely access to these heterogeneous subsystems through a uniform web service interface.

PACS: 07.05.Bx; 07.05.Dz

Keywords: Data acquisition system; Configuration; Control; Testing; Hardware; Extensible markup language (XML)

*Corresponding author. CERN, Geneva, Switzerland. Tel.: +41 22 767 4227; fax: +41 22 767 8940.
E-mail address: ildefons.magrans@cern.ch (I. Magrans).

0168-9002/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.nima.2005.03.034

1. Introduction

An important subsystem of the data acquisition system (DAQ) for the CMS experiment [1] is the hardware management system. This is in charge of the configuration, control, and testing of the DAQ hardware modules: Complex sequences of initialization and parameterization of the hardware must be defined and retrieved through available databases. These sequences, for instance, set tunable parameters like threshold values or parameters to compensate the accrued radiation damage. Numerous distinct hardware modules are replicated and common procedures for configuration are required. However, configuration is only part of the problem: a mechanism to execute tests on hardware devices and for detecting and diagnosing faults is needed.

The design of the DAQ hardware management system for the CMS experiment should cope with the following complexity dimensions:

1. *Number*: O(10^3) hardware modules.
2. *Evolution*: The preparation and operation of high-energy physics experiments typically spans over a period of many years (i.e. 1992, letter of intent for the CMS experiment [2]). During this time, the hardware and software environments evolve towards the operational phase.
3. *Human*: despite the necessary and highly hierarchic structure in a collaboration of more than 2000 people, different subgroups might implement solutions based on heterogeneous platforms and interfaces. Therefore, this would increase the cost of design and maintenance of a central control system.

We believe that a software environment based on the Xseq (Cross-platform sequencer, or extensible markup language (XML)-based sequencer) model [3] would simplify the configuration management of these software systems. Xseq is based on a comprehensive use of XML to describe:

1. *Configuration, control and test sequences through a new high-level XML-based programming language (Xseq)*: This offers a common set of functionalities, the core package, that cope with frequently encountered use cases in data acquisition environments. The core package can be extended, following a modular approach, by means of the XML schema [4] technology in order to cope with the specific requirements and to optimize execution time.
2. *Associated data*: Device descriptions, configuration data and execution reports.

2. Software environment: Xseq

A software environment (see Fig. 1) designed following this model has been implemented [3]. This comprise: xsd documents that define the syntax of control sequences and associated data, examples for every feature of the language, a user manual and language reference, and an interpreter for Xseq programs implemented in C++ under Linux available as a standalone tool and as a pluggable module for a distributed programming framework, specifically designed for data acquisition in the CMS experiment (XDAQ [5]).

2.1. How to

Xseq is an interpreted language that offers: both imperative and object-oriented programming styles, local execution of remote sequences with parameter passing by reference, separated syntax and semantic language extensions, scoped non-typed variables with automatic garbage collection, embedded validated XML documents, automatic and controlled creation of execution reports, and a remote procedure call mechanism based on SOAP [6] messages.

Additional functionalities have been added to the core syntax in the form of modular XML schema extensions, in order to fit frequently encountered use cases in data acquisition environments: transparent access to PCI and VME devices, file system access, SOAP messaging, DOM [7] and XPath [8] interface, and system command execution interface with redirected standard error and standard output to internal string objects.

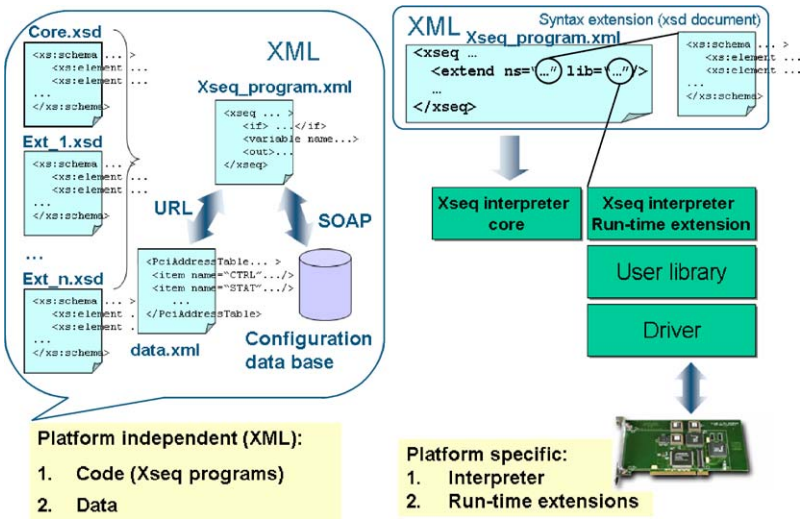

Fig. 1. Software environment parts: This comprises the core syntax of the language (Xseq) and associated data defined as xsd documents, and a platform-dependent interpreter available as a standalone tool and as a pluggable module for a distributed programming framework.

In Fig. 2, an example is given of how the hardware access is performed. Device specifications, configuration data and control sequences are XML documents. In this example, configuration data are retrieved through an XPath query from a configuration database. The language syntax extensions are declared in the first command of the programa ($\langle xseq \rangle$). Each syntactic extension is associated with an interpreter run-time extension by means of the Xseq command $\langle extend \rangle$. This facility makes it possible to separate syntactic language extensions, defined in XML schema modules, from the run-time interpreter extensions. Therefore, different implementations of a given extension can be mapped to one syntactic language extension. The possibility to keep a constant syntax while the interpreter run-time extension is modified reduces the need to modify the code as the underlying subsystems are changed. It also allows resource sharing among different subsystems with similar requirements.

Every object has an optional attribute *report*. When switched on, the object reports all operations on it. The execution report is serialized into a XML document, by means of the command $\langle get\_report \rangle$. This facility helps to automatically generate hardware test reports that can be validated against a given xsd document.

Xseq supports remote procedure calls for executing code on other machines. The remote procedure call mechanism is used together with the pluggable XDAQ interpreter module. The optional attribute *rpc_msg* is used when an Xseq program is remotely executed. *Rpc_msg* defines the name of the variable that contains the SOAP message used to remotely execute the script. The SOAP message itself can be processed inside the script using the language extension to manipulate SOAP messages. This functionality is useful when parameters must be remotely passed. Finally, every Xseq program ends by returning a SOAP message that will be forwarded to the client.

The uniform use of XML to define configuration data, device descriptions, execution reports and code facilitates the automatic and consistent generation of documentation artifacts by means of XSLT transformations [9].

3. Application scenarios

The section presents some application scenarios carried out for the CMS experiment. First, we

Fig. 2. Program example in Xseq language.

introduce some hardware modules belonging to the First-Level Global Trigger [10], the Silicon-Tracker Sub-detector and the DAS [11]. Different hardware management aspects, and methodology details bound to the model are demonstrated. In the second part, all application scenarios have been integrated and a large hardware management system has been implemented.

3.1. Standalone applications

- A testing system for the GIII [12] showed how already compiled applications can be integrated by using the Xseq language extension that permits one to execute system commands and redirect standard output and standard error to internal string objects. They can then be serialized to XML documents or transmitted through SOAP messages. This hardware module has been used also to prove that low-level drivers can be written with Xseq using the language extension to access to PCI devices. Therefore, a fine granularity of control is possible.
- A set of Xseq scripts to test memory modules has been written. The same scripts have been used to test a PCI and VME RAM modules, and a PCI flash memory. Using one script for many modules demonstrates how this approach eases system evolution and resource sharing among subsystems. The execution report of the memory test is automatically generated and serialized into a XML document. This can be validated against a given xsd document creating standardized execution and debugging documentation.

- A configuration and control system for the First Level Global Trigger has been developed. This exemplifies how the core package can be extended in order to integrate a user-specific C++ API as a customized language extension. This extension has been documented, and is a proposal for a methodology to integrate specific subsystem requirements, maximizing future code reuse, foreseeing a future hardware platform evolution. In order to fit well with the distributed nature of the CMS DAS, the pluggable version of the interpreter, compatible with XDAQ, has been used. With this architecture, all configuration and control sequences are available as web services accessible through a SOAP interface. The approach of using web services technology to manage the data acquisition hardware devices homogenizes both: hardware and software configuration and control interfaces. This simplifies the design and implementation of a configuration and control system. In addition, the comprehensive usage of the XML syntax eases a distributed storage of code and data.
- *Fast merging module* [13]: a configuration and testing system for this TINY-based module has been implemented. This shows how the independence between syntactic and interpreter extensions eases the future evolution to CompactPCI technology. In this case, the scripts can be reused with only little modifications, i.e. the link to the register description file.
- *I2C chain for the Silicon-Tracker Sub-detector*: the separation between syntactic and interpreter run-time extension has allowed the design of a specific extension that would ease the migration from a customized driver to a future integration with a common hardware access library (i.e. HAL [14]).

3.2. A hardware management system supervisor

All application scenarios have been integrated and a large hardware management system has been implemented (see Fig. 3). This is in charge to buffer all calls from clients and to forward the responses from the different hardware management subsystems to the clients. The RPC mechanism, based on SOAP messages, together with the pluggable module of the interpreter for XDAQ has been used.

4. Summary

A Software environment to describe configuration, control and test systems for data acquisition

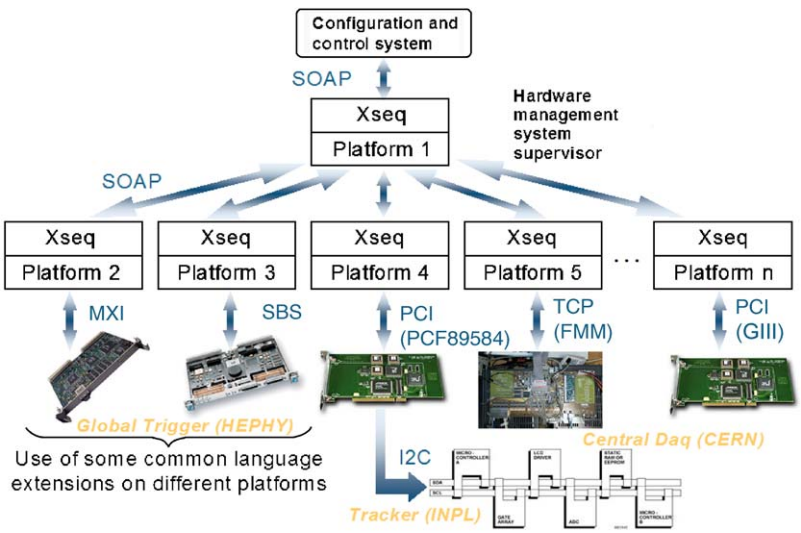

Fig. 3. Hardware management system supervisor based on the Xseq software environment.

hardware devices has been presented. The design follows a model that enforces a comprehensive use of XML syntax to describe both the code and associated data. This environment includes: xsd documents that define the syntax of control sequences and associated data, examples for every feature of a new XML-based language, a user manual and language reference, and an interpreter, implemented in C++ under Linux, available as a standalone tool and as a pluggable module for a distributed programming framework, specifically designed for data acquisition in the CMS experiment (XDAQ).

A number of application scenarios, for the CMS experiment, have been outlined. We introduced some hardware modules belonging to the First-Level Global Trigger, the Silicon-Tracker sub-detector and the Data Acquisition System (DAQ), and for each of them different hardware management aspects and methodology details bound to the model have been presented. Finally, the above subsystems were integrated into a large hardware management system. It has been shown how a common layer, based on this software environment, to manage heterogeneous DAQ hardware modules, simplifies the work of a centralized management system and eases configuration management by easing resource sharing, platform evolution, and automatic and consistent maintenance of documentation artifacts.

References

[1] The CMS Collaboration, The Compact Muon Solenoid, CERN, Technical Proposal, No. 7, LHCC 94-38 December 1995.
[2] The CMS Collaboration, The Compact Muon Solenoid, Lettter of Intent, CERN/LHCC 1992-3.
[3] I. Magrans, et al., Uniform Approach Based on XML Technologies to Manage Data Acquisition Hardware Devices, NSS-MIC 2003, Portland, Oregon 19–25 October 2003.
[4] XML Schema. W3C. [Online]. Available: http://www.w3.org/XML/Schema, 2004.
[5] J. Gutleber, L. Orsini, Cluster Computing vol. 5(1), Kluwer Academic Publishers, Dordrecht, 2002, pp. 55–65.
[6] Simple Object Access Protocol (SOAP). W3C. [Online]. Available: http://www.w3.org/TR/SOAP, 2004.
[7] Document Object Model (DOM). W3C. [Online]. Available: http://www.w3.org/DOM/, 2004.
[8] XML Path Language (XPath). W3C. [Online]. Available: http://www.w3.org/TR/xpath, 2004.
[9] XSL Transformations (XSLT). W3C. [Online]. Available: 2004
[10] C.-E. Wulz, Nucl. Instr. and Meth. A473/3, 231.
[11] The CMS Collaboration, The Trigger and Data Acquisition Project, volume II, Data Acquisition and High-Level Trigger, Technical Design Report, CERN/LHCC 2002-26, CMS TDR 6.2, 15 December 2002.
[12] E. Cano, et al., FED-kit design for CMS DAQ system, in: Proceedings of the Eighth Workshop on Electronics for LHC Experiments, Colmar, France, 9–13 September 2002, pp. 274–277.
[13] E. Cano, et al., The fast merging module (FMM) for readout status processing in CMS DAQ, Ninth Workshop on Electronics for LHC Experiments, Amsterdam, 29 September–3 October 2003.
[14] [Online]: http://cmsdoc.cern.ch/cms/TRIDAS/html/Documents.html, 2004.