ASSET MANAGEMENT APPLICATION FOR LLRF CONTROL SYSTEM

B. Sakowicz*, M.Kamiński, D. Makowski, A. Piotrowski, P. Mazur, A. Napieralski, DMCS TUL, Poland

Abstract

This paper presents the base assumptions of a LLRF Asset Management System [5]. The application consists of several database modules which are designed to facilitate management and operation of a distributed control system composed of many hardware and firmware nodes, and spread geographically across a wide area.

The system is currently deployed in DESY institute in Hamburg, Germany to facilitate tasks of managing dynamically changing LLRF system used in FLASH and XFEL [1, 2, 3] particle accelerators.

INTRODUCTION

A digital LLRF (Low Level Radio Frequency) control system is currently used as a part of many accelerators, which include the X-Ray Free Electron Laser (XFEL) currently located in Hamburg, Germany.

The control system was developed to provide high availability and reliability, therefore it was decided to incorporate the ATCA (Advanced Telecommunication Computer Architecture) and AMC (Advanced Mezzaine Card) standards. These standards also provide a module oriented design of the system, and provide several levels of redundancy for key subsystems such as power management, diagnosis and communication.

The system is currently composed of few dozens ATCA stations built with an application of FPGA (Field Programming Gate Array) devices. Each ATCA station cat contain multiple FPGA devices, and every FPGA device needs to be programmed using a binary firmware file that contain the FPGA matrix configurations, binary files for PROM (Programmable Read Only Memory) and BDSL files (Boundary-Scan Description Language) that contain descriptions of the JTAG chain of these devices. The overall structure of the system is presented in Fig.1.

APPLICATION

The application design started on the basis of a simple firmware database application to facilitate management of the binary files used in a LLRF system [6]. In time the idea evolved into a much more complex solution which could provide the following functionalities:

- Store and modify the current structure of most of the devices comprising the accelerator
- Store the DOOCS[1] parameter types and values used by these devices
- Store and retrieve information about cable wirings in the accelerator structures
- Store firmware binary and source files for each FPGA board present on the system
- Provide information about signals being generated by devices, including persons responsible for any particular system

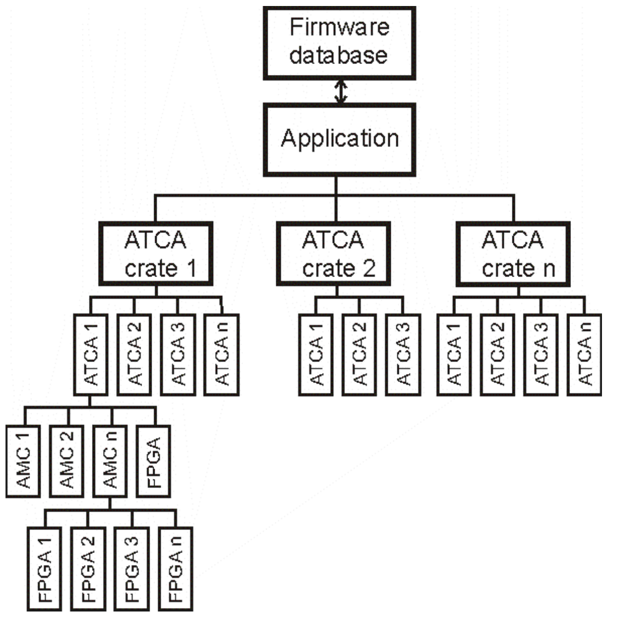

Figure 1: An example structure of the firmware database nodes in the FLASH accelerator.

DESIGN

In order to describe the internal structure of components used in LLRF system, a graph based approach was taken into consideration. LLRF components such as:

- Accelerator hardware elements
- Internal wiring between components
- Accelerator software elements

may be described in a form that resembles a tree graph which contains cycles. Core LLRF elements may be presented in a tree-form alone, but considering the wiring between these elements and other relations which may arise at a later time inclusion of cycles to the data model was necessary.

The system was developed using the J2EE platform, utilising only open source libraries such as:

- Hibernate Framework as an ORM solution
- Spring Framework used as an IoC container [4]
- Struts 2 framework as an MVC Controller
- Apache CXF framework for providing Web Service access

*{Sakowicz, kaminski, makowski, komam, pmazur, napier}@dmcs.pl

[1] DOOCS – Distributed Object Oriented Control System – software system for storage and retrieval of objects within an distributed environment.

Copyright © 2011 by PAC'11 OC/IEEE — cc Creative Commons Attribution 3.0 (CC BY 3.0)

The data was therefore described as an tree-like structure comprised of classes written in Java language. In order to represent the node (device, firmware etc..) hierarchy the inheritance of data structures was used. Each node type in the tree structure is modelled as a Java class file, having a set of pre determined fields for example: node name and description, and also a set of new fields which are particular for that specific node for example: a list of inputs and outputs for devices supporting cable wiring, or a list of binary attachments for firmware and documentation storage. The node classes can then be extended to introduce new node types based on other node types.

The model of the single node of the structure is depicted in Fig. 2.

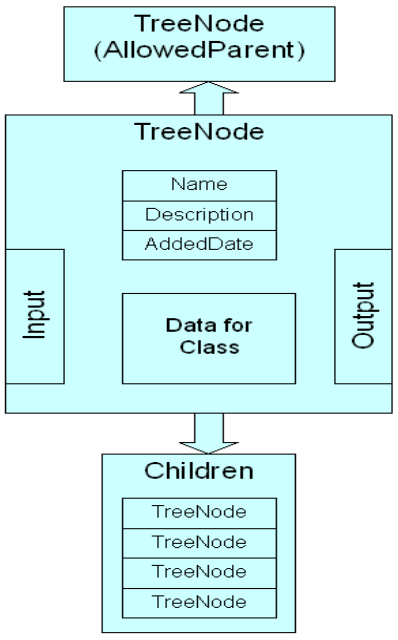

Figure 2: Java Class representation of the tree structure.

DATA STORAGE

The resulted data model was then described as a set of entities of the Hibernate Java framework using the “single table” inheritance strategy. This allows to map an complex set of similar Java classes to a single table. Therefore the core tree node data is represented in the database with only one, single database table containing columns for all available fields in every node. The resulting model may seem redundant at first, as it produces wide database rows, but it also facilitates database management and allows for performing more advanced node operations such as searching based on complex criteria using much less complicated database queries. The performance gain of using the “single table” inheritance strategy is therefore clearly visible.

In order to maintain internal data consistency a set of constraints must also be set. Any node present in the resulting tree should be subjected to following constraints.

- Structure constraints: ensuring that a specific type of node may be inserted only in a particular place in the tree structure
- Field constraints: ensuring that the specific node has its fields correctly filled out.
- Additional constraints: for example: enforcing other constraints which affect the structure of the device tree. For example: an FPGA board can only be inserted into a FPGA crate which supports the same model of board

Most of these constraints were modelled using features already available in the Java programming language. The inheritance of node classes allowed to define an single point of defining properties that influence the whole device tree. For example: adding an constant list of allowed parent node types to the specified node type can result in satisfying the first structure constrain. When an end user is adding new tree node into the structure (for example a new FPGA device to ATCA crate), the application checks if the ATCA device node can be a parent of the FPGA device node.

Additional constraints were satisfied using the Hibernate Validation framework with the use of JSR-303[2] Java Annotations to simplify the validation layer of the application.

USER INTERFACE

The application user interface was implemented as a web page using Apache Struts 2 framework. To increase the interface response time application depends heavily on AJAX[3] calls, which provided a significant interface performance gain and allows for convenient tree navigation directly from a web page.

SIGNALS DATABASE

The asset management system also include features which facilitate management of the overall structure of the accelerator. The signals database comprises of a set of “signal” and “function” nodes which can be added to any other node in the device structure.

Signals which are added to the specified node represent actions which this node can generate. Signals are then being used by functions which can also be assigned to any node in the system. Functions are designed to transform its input signals ie: take one signal as the input signal and return other signal as the output.

The structure validation process ensures that every signal present within the tree structure which is being used as an output signal of any function is also being used as an input signal by some other function. Therefore it

[2] JSR-303 – Java Specification Bean Validation standard.

[3] AJAX – Asynchronous Javascript And XML – technology used to dynamically update fragments of a web page, using asynchronously fetched content.

Copyright © 2011 by PAC'11 OC/IEEE — cc Creative Commons Attribution 3.0 (CC BY 3.0)

assures that every signal present in the database is actually needed by some other process.

In addition signals database may also be used as a mean to report work progress on any particular activity (an activity is represented as a signal).

OTHER FEATURES

Some of the application features, mostly concerning firmware management, were also exposed as Web Services in order to integrate with existing software used to program devices present in the accelerator structure.

To provide firmware programming capability directly from other machines present within the DESY network an Java client software has been also developed. The client software uses the Web Service Interface exposed by the asset management application to provide means for selecting the desired firmware to be loaded into FPGA device and initiate the firmware loading process. The client application also communicates with one of the DOOCS servers in order to manage the firmware loading stage.

To address performance issues present especially during start-up and initialization phase of the accelerator when hundreds of devices need to be programmed simultaneously, a local caching mechanism was used, that stores the currently used firmware on the device storage itself.

CONCLUSIONS

The asset management system application is currently being tested to manage hardware and software components that are and will be present in the FLASH and XFEL accelerators. It offers a significant improvement in firmware management time and can be used to keep track of changes within the device structure of the accelerator and its internal wiring connections.

In addition, separate management modules were included into the application which provide validation of the device structure model by means of using Signals and Functions.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the Polish National Science Council Grant 642/N-TESLAXFEL/09/2010/0.

The authors are scholarship holders of the project entitled ”Innovative education ...” supported by European Social Fund.

REFERENCES

[1] A. Schwarz., “The European X-Ray free electron laser project at DESY”. 26th Inter-national Free-Electron Laser Conference, pages 85–89, August 2004..

[2] Deutsche Elektronen-Synchrotron. “X-Ray Free Electron Laser”, Interim Report of the Scientific and Technical Issues (XFEL-STI) Working Group on a European XFEL Facility in Hamburg. Technical report, Deutsche Elektronen-Synchrotron, January 2005.

[3] R. Brinkmann, K. Flottmann, J. Rossbach, P. Schmuser, N. Walker, and H. Weise., Technical design report, PART II-The accelerator. Deutsche Elektronen-Synchrotron DESY, 2001.

[4] Johnson, Rod; Jürgen Höller, Alef Arendsen, Thomas Risberg, and Colin Sampaleanu (2005). Professional Java Development with the Spring Framework. Wiley. ISBN 0-7645-7483-3.

[5] S. N. Simrock, “Low level radio frequency control system for the european XFEL,” in Mixed Design of Integrated Circuits and System,2006. MIXDES 2006. Proceedings of the International Conference, Jun. 2006, pp. 79–84.

[6] Kamiński M., Makowski D., Mazur P., Murlewski J., Sakowicz B.: "Firmware application for LLRF control system based on the Enterprise Service Bus", CADSM 2009, Ukraina, ISBN 978-966-2191-05-9

Copyright © 2011 by PAC'11 OC/IEEE — cc Creative Commons Attribution 3.0 (CC BY 3.0)