Debbie: an innovative approach for the web-based interface to the CMS Pixel Tracker configuration DB

Marco Rovere[1]
CERN
CH-1211 Genève 23, Switzerland
E-mail: marco.rovere@cern.ch

Dario Menasce
I.N.F.N. Milano- Bicocca
Piazza della Scienza 3, Edificio U2, Milano 20126, Italy
E-mail: dario.menasce@mib.infn.it

Abstract

The configuration of the CMS Pixel detector consists in a complex set of data that uniquely define its startup condition and the optimized calibration constants. Since several of these conditions are used to both calibrate the detector over time and to properly initialize it for a physics run, all these data have been collected in a suitably designed database for historical archival and retrieval. In this paper we present a description of the underlying database schema with a particular emphasis on the architecture and implementation of the web-based interface that allows for very sophisticated browsing/editing operations of detector data using a graphical representation of its topology. This interface employs state-of-art technology such as Ajax transactions, SVG-based vector graphics and an extensive use of the Extjs JavaScript library. The GUI represents a novel approach to web-based interfaces, since it features a very complex set of widgets, dynamically generated on the fly upon user-demand, thus mimicking the behavior of a stand-alone program specifically designed to this extent, but avoiding portability and interactive-login issues for the latter solution.

13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research - ACAT 2010
Jaipur, India
February 22–27 2010

[1] Speaker

Introduction

The CMS pixel detector is a rather complex device consisting of about 16k readout chips (ROCs) and several ancillary components, all of which must be properly initialized and eventually calibrated before a physics run can be taken. The configuration of a detector at any given time is described by suitable data structures stored in a database for retrieval and historic archival. In this paper we describe the interface to the CMS Pixel Detector Configuration Database, consisting in a set of server procedures and a web browser client capable of very sophisticated interactive manipulation of the data. Main feature of this client is the highly graphical and point-and-click nature of the navigation tools offered to the user. To this extent we employed state-of-the-art technologies such as Ajax transactions, SVG vector graphics and we adopted the Extjs JavaScript framework to display large amounts of data in very compact and efficient widgets.

1. The Pixel Detector

The Pixel Detector consists of two distinct components: a central barrel, made of 768 detector modules arranged into half-ladders of four identical modules each, and four forward disks made of 672 detector modules of seven different sizes arranged into blades. To read out the detector about 16 000 readout chips (ROCs) are bump-bonded to the detector modules. Each ROC features 26 programmable registers to control its behavior during calibration and readout: these data constitute part of the detector's condition dataset that will be described later. A complete description of the detector layout can be found elsewhere[1].

1.1 The detector hierarchy: physical and logical topology

A crucial requirement to define the status of the detector for calibration or data taking at any given time is the ability to define ***partitions***. A partition is a subset of the whole detector that needs special treatment (e.g. to be disconnected from readout). Given the large amount of components featured by the detector, it is often cumbersome and tedious to specify long lists of components to be partitioned. To vastly improve the efficiency in this kind of operation we designed our interface to leverage the natural hierarchical structure of the components that make up the detector. The basic building block of the pixel detector is the ROC. ROCs can be assembled together into larger components (plaquettes for the Forward and modules for the Barrel, see fig. 1) which, in turn, can be assembled in even larger components (such as blades for the Forward and Ladders for

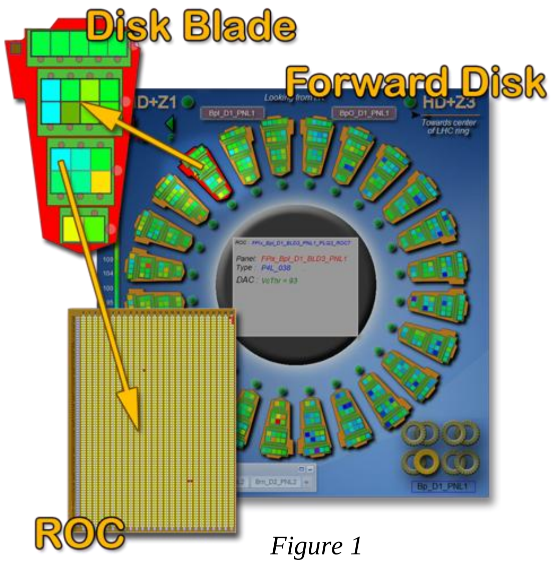


Figure 1

the Barrel). While this hierarchy reflects the physical layout of the detector, it is possible to build another hierarchical view based on the topology of hardware connections of the readout chain. Small groups of ROCs are connected to ancillary chips, called TBM, to synchronize the readout with the radiofrequency of the beam. These, in turn, are connected to opto-electronic components to transform the analog signal to be transmitted along fiber optic lines to the FED cards which constitute the last component in the readout chain under the responsibility of the pixel detector. In both cases, it is possible to define a partition by specifying the largest component in one of these two hierarchical descriptions to automatically include all sub-components. This behavior is complemented by a graphical point-and-click interface, described below, which allow users to specify arbitrarily complex partitions in a very small time.

2. The detector configuration dataset.

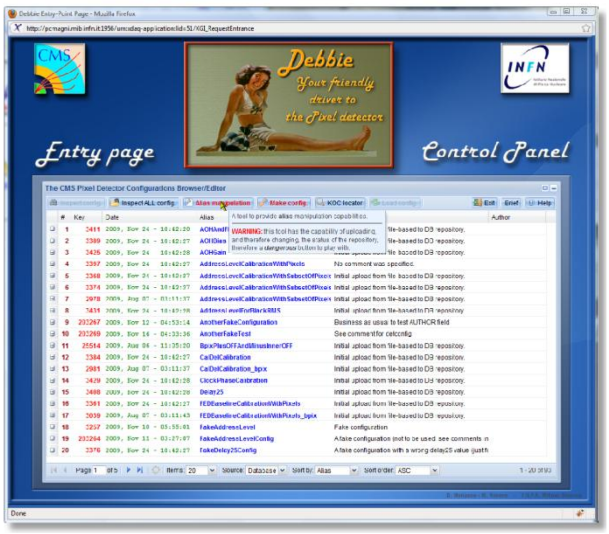
A complete description of the detector condition at any given point in time is called a ***configuration dataset***. Such a dataset is divided into several, distinct, components (KOCs, for Kind Of Conditions) each one encompassing different aspects of the status of the detector. One KOC describes the status of individual ROCs (whether they are included in the readout stream, biased, initialized and so on), another contains the table of the 26 registers that specify each ROC’s behaviour. A map of *topological* connections between electronic components is included as well, along with several other KOCs. It is beyond the scope of this paper to describe these datasets in detail, since their definition was assumed as a prerequisite of the design of our interface[2]. Crucial for the scope of this paper is to note that these KOC’s do not change all at the same time: the configuration of the detector at a given time is a mixture of KOC’s defined at different times. We will see that this implies sophisticated manipulation tools to allow users to specify the correct KOC recipe of a configuration. The current implementation of these configurations and their KOC components is a full-fledged ORACLE database.

3. The overall architecture

Debbie consists in three main components: an ORACLE database, a custom-made web server and a web client. We currently have two distinct databases running at CERN, a development and a production one, which is not discussed here. The web server is a C++ application, based upon the XDAQ[3] framework. This application listens for user requests issued from a web browser as Ajax[4] transactions and responds back with a payload of data selected from the DB repository. It basically acts as a broker between user requests and the ORACLE database: its main feature is the ability of formatting answers as XML or JSON data streams, suitable for immediate rendering by the web client which issued the request. During an interactive session, users navigate across detector components and visualize or change potentially large amounts of data. These data are cached locally in the server: only the limited amount of data needed to be rendered on the web client is actually transmitted back and forth between client and server, thus reducing network latency, the amount of memory allocated by the client and speeding up the overall navigation capability of the tool. Since this caching mechanism violates the state-less characteristics of the connection to a web-server, only one user at a time can connect to the server in read-write mode. A mechanism is in place to grant such an access for a limited time: before expiration users are prompted for eventual renewal of the grant. Other users can still connect meanwhile, but only in read mode, for monitoring purposes.

4. The navigation.

The entry page of Debbie consists in a single widget to allow a selection of the desired configuration to visualize or edit. This widget (see a snapshot in fig. 2) features all the necessary tools to explore, visualize, navigate and load the configurations stored in the database. An essential characteristic is the paging and filtering mechanism: since the list of available configuration is already large and will grow over time in the coming years, only a reduced number of configurations is displayed at any given time. When a user needs to display more, a custom made navigation bar at the bottom of the page allows to request a new page, go back to the previous or reach the first or last page. Another very useful and versatile tool for navigation is the server-side filtering system that users can apply to the values represented in each column of the spreadsheet-like table. To filter configurations based on their creation time, a specialized calendar allow users to specify a particular date or a time interval, as shown in fig. 3. For columns containing a string, such as the configuration alias, the filter consists in a user-specified regualar expression. This is very powerful since it provides a very fine-grained tuning of filtering capabilities. Last but not least, for numerical columns, such as the configuration number, the filter consists in a numerical value or a range of values. Particular care has been taken to provide users with a compact, yet intuitive navigation tool. Once a user has chosen the configuration he would like to work with, several buttons allow for a preliminary visualization or manipulation: users can inspect the definition of the chosen configuration in terms of its KOC components, a different configuration can be selected searching for a specific combination of KOCs or by searching where a particular KOC is referenced. Each of these tools pops up a specific widget with the necessary elements to drive the selection. When a configuration is requested, the server contacts the DB and pre-loads in its memory the minimum amount of information needed to start a point-and-click navigation across the detector. As



Debbie
Your friendly driver to the Pixel detector
Entry page
Control Panel
INFN
The CMS Pixel Detector Configurations Browser/Editor

Figure 2

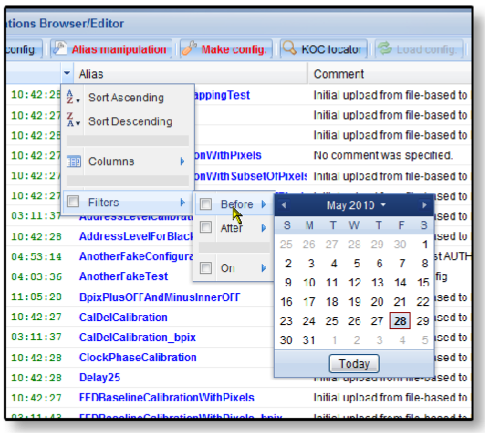


Figure 3

shown in fig. 4, a special emphasis is placed upon the purely graphical representation of the detector. This allows users to

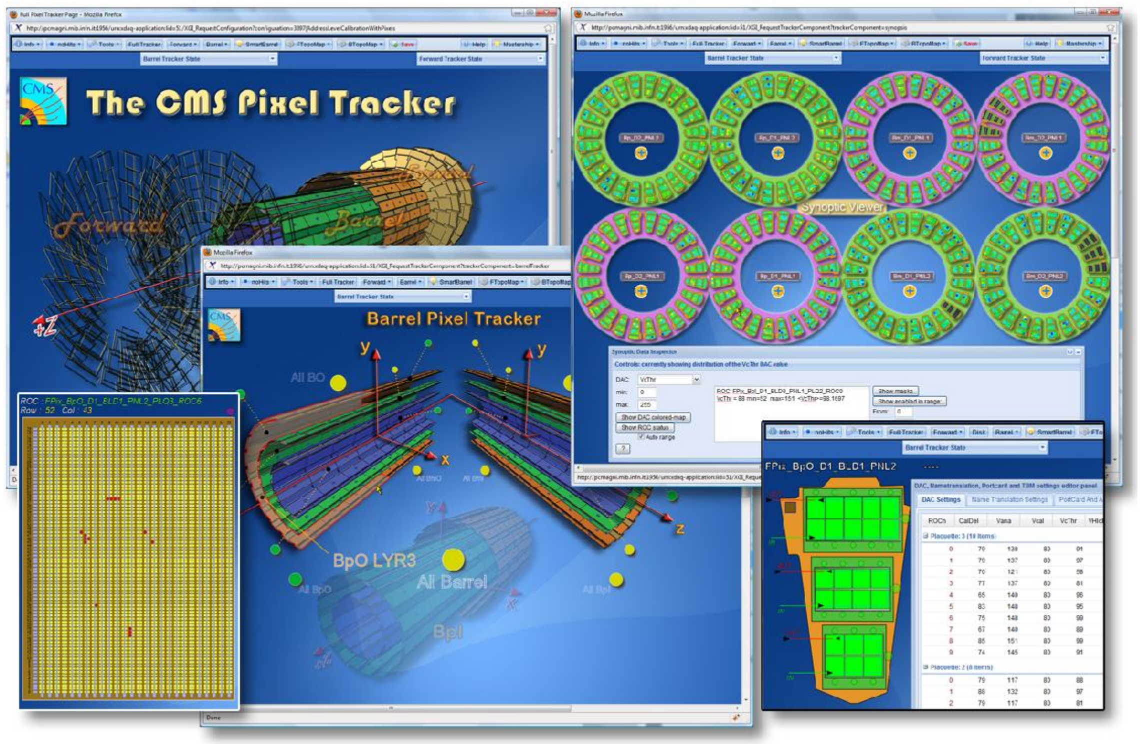


Figure 4: a collection of snapshots taken during navigation across the detector's components

navigate across components by just point and click. There are exceptions to this behaviour of course: to allow particularly sophisticated partitioning schemes of the detector (such as disabling specific ROCs from the readout chain), auxiliary tools are provided. One such tool is once again a regular expression filter to select specific groups of components based upon a predetermined pattern.

5. Technical details

As mentioned in the introduction, the tool consists of a server, implemented in C++ in the context of the XDAQ framework, and a client, running on a web browser, implemented in a mixture of XHTML, SVG and JavaScript code. Additionally we rely upon an additional infrastructure, implemented by the FNAL group and programmed in Java, which is responsible of the loading of the XML-formatted data directly into the ORACLE DB. The server is able to listen for user requests on a specified port: several servers can run simultaneously on different ports, each one capable of handling requests from several clients in read mode and one client in write mode. Each user request ends up providing back to the originating client a payload of specially formatted data in XML or JSON format. When the request results in a graphical element, this is implemented as an SVG[5] payload, created by the server with the appropriate call backs for interactive behaviour. An SVG content is formatted as an XML DOM and allows for the SVG interpreter of a browser to be parsed, rendered and provide interactive context for user. Therefore a JavaScript snippet, sent along with the SVG payload, can provide all the necessary mechanisms for user navigation. Wherever a histogram is requested, the server provides the corresponding file (formatted as a png file) and the client renders the plot in an appropriate widget build with EXTJS components. The backend for histogramming is the ROOT[6] library.

6. Conclusions

A Pixel Configuration DB browser has been developed to aid users managing the vast repository of the CMS Pixel Configurations. A client-server architecture is the back-bone of the system, with a web-server in charge of handling users request to the configuration database and a client, a regular web browser such as Firefox capable of executing complex scripts provided by the server both as JavaScript snippets or SVG-based graphics is in charge of handling the graphical representation of the detector and responding to user input. The functionality is complemented by an ORACLE based backend. Central to the design of the system are the concepts of *point-and-click navigation* as well as *selection of components through regular expressions*. The latter allow users to define arbitrarily complex partitioning of components for tasks such as calibration or initialization of a subsample of the whole detector The detector complexity is broken down in two possible views, a *geometrical* one, where emphasis is on the physical location of components in space, and a *topological* one, where emphasis is on the logical connection of electronic readout components. The system is currently in use at the P5 area, next to the CMS detector, for regular every day use by personnel on shift.

References

[1] ***The CMS experiment at the CERN LHC,*** *2008 JINST 3 S08004* (pag. 33)

[2] A. Ryd, K. Ecklund et al, ***The Configuration DB interface***, Pixel Collaboration Internal Note

[3] J. Gutleber, L. Orsini, ***XDAQ, a platform for the development of distributed data acquisition system,*** https://svnweb.cern.ch/trac/cmsos

[4] J.J. Garrett**,** ***Ajax: A New Approach to Web Applications,*** http://www.adaptivepath.com/ideas/essays/archives/000385.php

[5] SVG: **Scalable Vector Graphics** http://www.w3.org/Graphics/SVG/

[6] ROOT: http://root.cern.ch/drupal/