Summer Student Report

Felix Wieland

August 27, 2014

The "Webmonitor" project I worked on supervised by Dr. Flavio Archili is aiming to set up of a web based data quality monitor. In this report I am briefly going to outline the goals of data quality management, give reasons for developing a new data quality monitor system and eventually describe the progress made during my eight weeks of stay at CERN.

1 The goals of DQM

DQM is a crucial part of the data taking during high energy physics experiments, ensuring the correct functioning of the measurement apparatus and thus laying the foundation for further successful analysis. During the process of data taking, the data quality shifter therefore selects certain plots to be compared with given references and receives a direct visual feedback through overlaying both plots.

Moreover, the DQM provides also some quantitative informations that can be used to better evaluate the quality of the data analysed. Given the complexity of the detector and its time evolution the monitoring process cannot be completely automatised and it requires the human supervision. However, modern data analysis techniques like machine learning might provide a useful toolset for further simplification.

In order to optimise the usability of the application, one strives to ensure compatibilty with existing databases, short loading times and an intuitive user interface, of which the design should be seperated from the actual programme code and logic. Furthermore, the application's code should be well documented and easily maintainable as well as scalable based on LHCb's future needs. Histograms should be automatically read out from their corresponding ROOT Files on CASTOR using the Bookkeeping database informations to construct the file's exact location.

2 Reasons for a new DQM System

The main goal of the new DQM System is to reduce the effort for data quality shifters and also to reduce the time necessary to train prospective data quality shifters, which

bears considerable importance in big collaborations like LHCb. Furthermore, the new programme requires just an up-to-date browser (which makes it independent on the end user's operating system), a working internet connection and access CERN's webservices, in contrast to the existing DQM software (called Presenter) relying on ROOT and C++ and requiring the correct manual set up of the configuration file. Thus the Webmonitor allows for a less precipitous learning curve and for people from all over the world to easily collaborate directly from their institutes.

3 The state of development

Figure 1 and 2 depict the structure of the Webmonitor programme up to the present. On the server side, incoming HTTP request are being dealt with by a Python server relying on the Flask framework used together with the Jinja2 template system to allow for a separation of programme logic and code and user interface design.

The structure of the pages are loaded from the HistoDB and is then returned in JSON format to the browser to be displayed in a tree structure on the client's side. Each time the client takes action, (i.e. opens or closes a node, or clicks on the "open all" or "close all" buttons) the state of the tree is preserved on the server's side, so that, when the user reloads the page or logs in the next time, he can continue with working on the workspace he had left. Furthermore the caching of the tree on the server's side also reduces the amount of traffic to the database, and therefore speeds up the application. However, at each time the user can enforce a reload of the menu structure from the database, which can be important, if the page structure is altered at the same time.

Once the user has entered the run number, this number is sent to the server via an AJAX request and the programme automatically looks for all possible reconstruction versions for this run number from the BookkeepingDB, returns them and in the client a drop down menu is generated. If the last reconstruction version is still valid for this run number, this one is automatically selected and the corresponding files are read out to improve the usability of the programme.

The bookkeepingDB still requires lhcb-proxy-init to be called each time before the server starts, which could possibly be automatised by generating the proxy using an host certificate on the server machine.

As soon as the user selects one of the entries of the reconstruction version dropbox, the selection is sent again via an AJAX request to the server, which then creates the paths to the ROOT file on CASTOR and to the reference file on AFS and stores them locally, so that, when the user logs out the last working environment is again preserved.

When the user opens a tree node, the contents of the corresponding page are read out and the server returns a HTML page with stubs for each histogram, which are then filled in asynchronously by an AJAX request to the server, that reads out with pyROOT the histograms and returns the content in JSON format. A plugin to d3 called d3.chart is used to display the charts in the browser as an svg object. The histograms stored on CASTOR cannot be read out by the user and therefore this task is left to the server in contrast to the histogram plotting conducted in the client's browser, which represents a

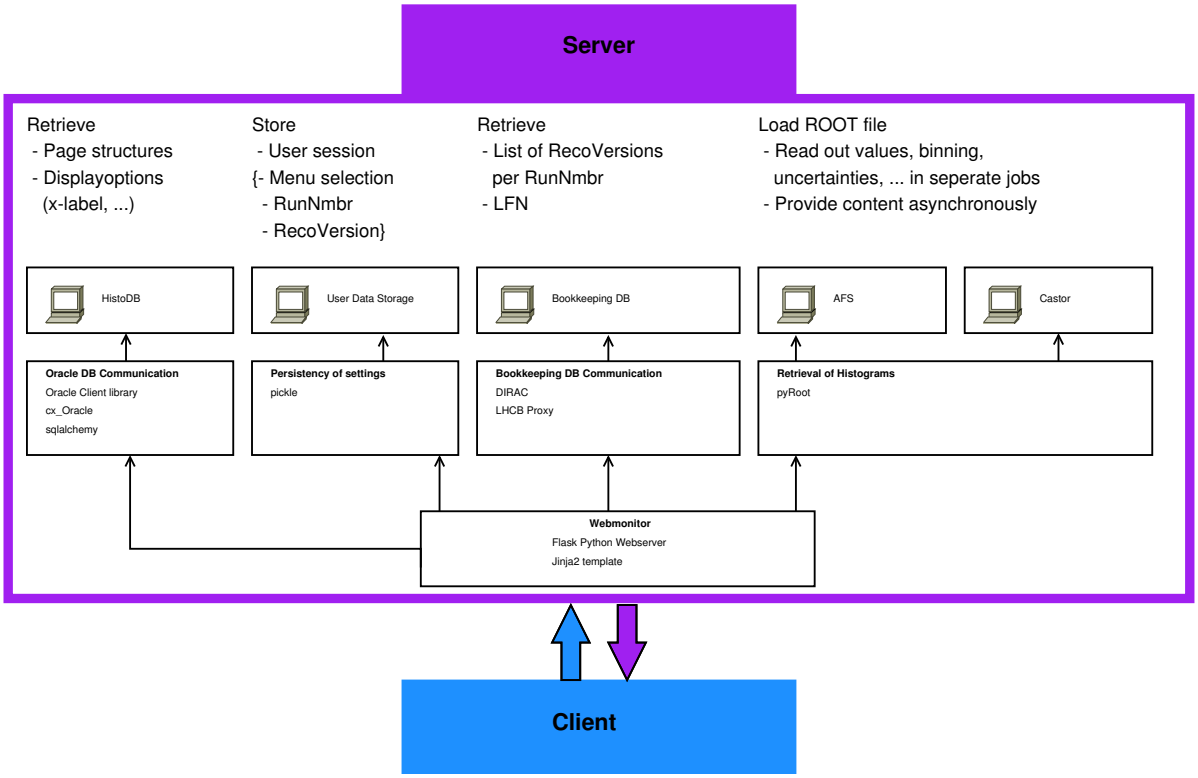


Figure 1: Diagram of the server side, outlining the different databases, packages and toolsets used in the program.

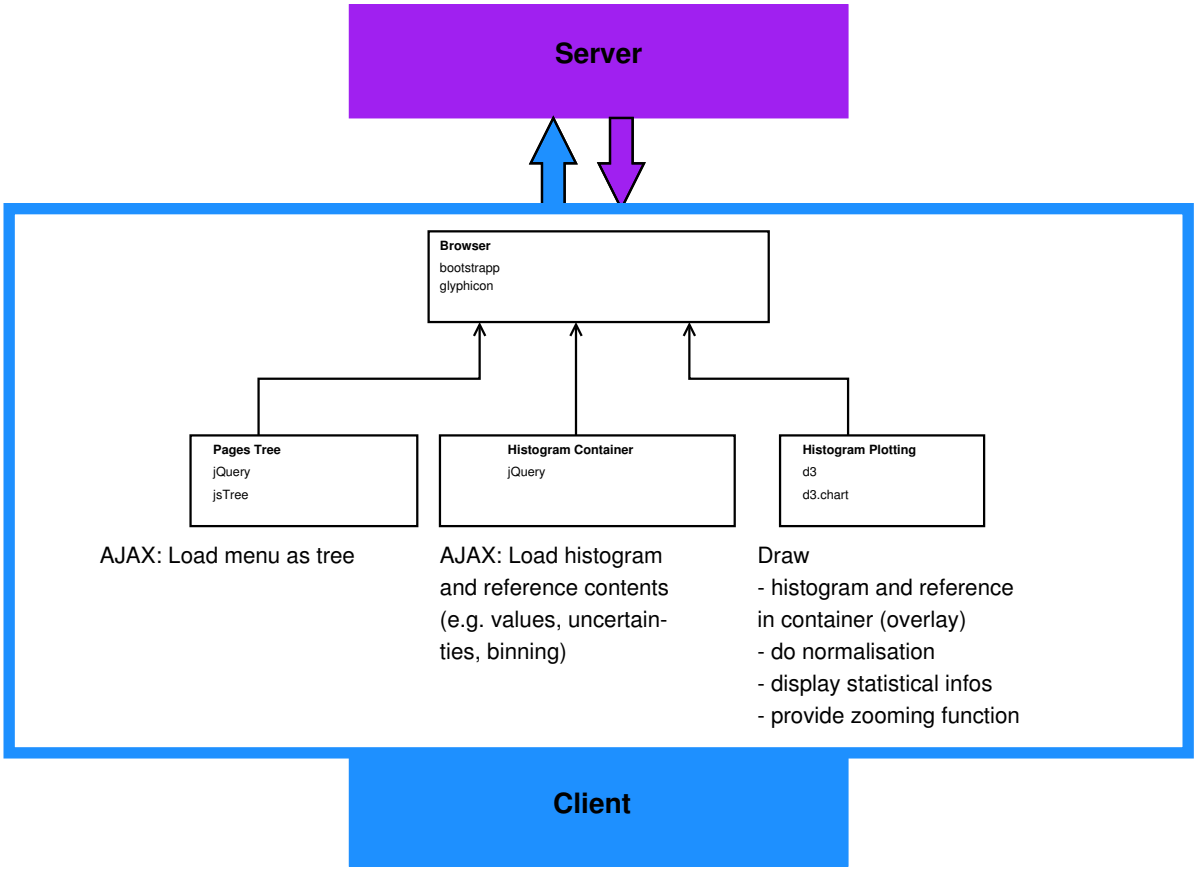


Figure 2: Diagram of the client side, outlining the different packages and toolsets used to display the program in the browser.

compromise, because it reduces the server load.

The user can also export plots to PNG format on his hard drive and can switch between the reference and the non-reference mode differentiating in the overlay of the reference plot in red over the actual measured data.

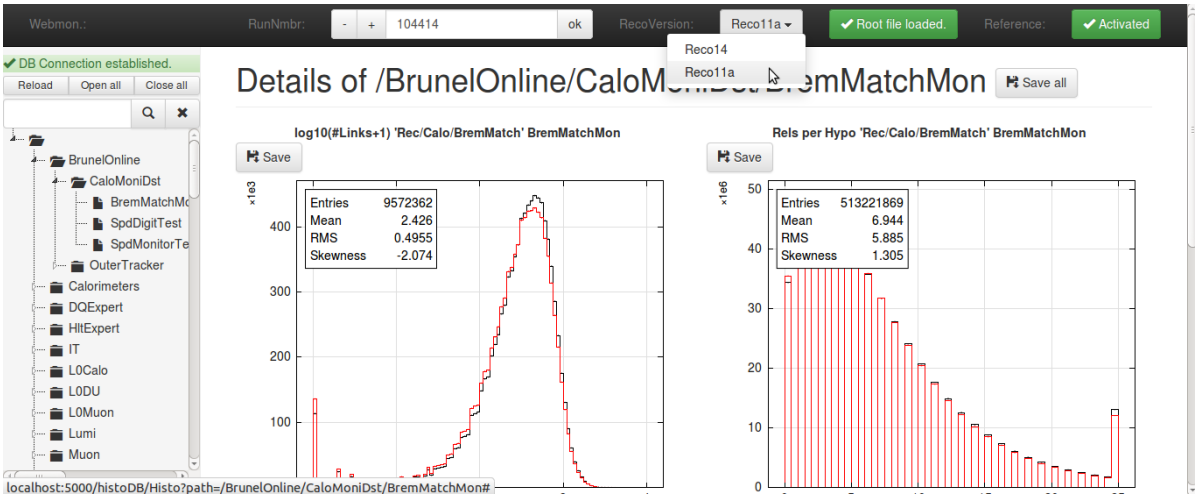


Figure 3: A screenshot of the running application showing histograms and their references.

A basic errorhandling has been implemented consisting of an errorhandling class encapsulating the logging and display error functions and a status button within the top bar of the web page, giving the user direct visual feedback, e.g. if the ROOT file was loaded succesfully.

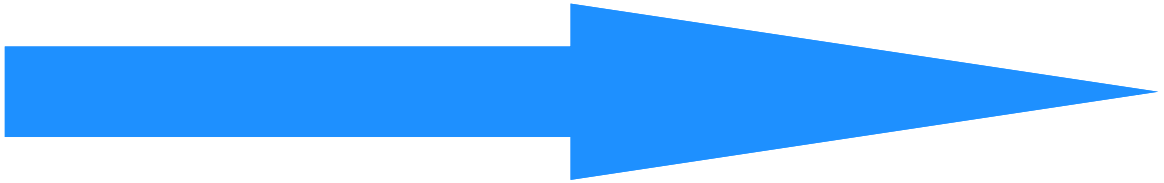
The last page contains a diagram listing the typical actions by the user and the following communication between server and client.

4 What I have learned

From this project I gained an insight into

- the process and importance of DQM
- ROOT and pyROOT
- modern web page development with Flask, Sqlalchemy and jQuery
- HTML5
- how experimental data can be organised in a efficient way
- how to access different file systems
- how databases can be used to store information and metadata about histograms

Work flow for the user



User initiates HTTP request in the browser

Server generates Overview page containing
- container for tree structure of pages
- input fields for runNmbr and recoVersion

Browser sends AJAX request to the server to ask for the menu

Server retrieves menu structure either
- from a local cache
- from HISTODB

based on the user's request and returns structure in JSON format

Browser builds tree with jsTree and sets up the rest of the Overview page

Browser builds tree with jsTree and sets up the rest of the Overview page

User enters runNmbr

Browser sends AJAX request to the server, specifying the runNmbr

Server stores runNmbr and retrieves from BookkeepingDB all possible recoVersions for this runNmbr and sends the data in JSON format

Browser creates dropdown menu of possible recoVersions

User chooses recoVersion

Browser sends AJAX request to the server, specifying the recoVersion

Server stores the recoVersion and retrieves the TCK corresponding to the runNmbr and the recoVersion from BookkeepingDB.
After that the correct ROOT files for the histograms of the run and of the run and of the reference looked for and their path's stored locally.

User chooses certain page in the menu tree

Browser sends AJAX request to the server, specifying the tree state.

Server stores the tree state locally to allow for the user to have the same folders openend when reloading or changing the displayed page

Browser initiates a HTTP request to the Histo-Viewer, listing the histograms in the page selected in the tree menu.

Server retrives the histograms to be displayed and the corresponding metadata (name, axis labels, ...) from histoDB and generates a HTML page with empty containers for the histograms.

Browser initiates an AJAX request to the server for each histogram container asking for the actual data (values, binning, ...). This allows for an asynchronous loading and a better user experience

Server retrives the histograms to be displayed individually, reads them out with pyROOT and returns the data in JSON format. Histograms are read from CASTOR, References are read from AFS

Browser draws the data with the d3.chart library and normalises as well (if set) the references. A fucntionality to save the histos as PNGs is provided.