THE CERN BEAM INSTRUMENTATION GROUP OFFLINE ANALYSIS FRAMEWORK

B.Kolad, J-J Gras, S. Jackson, S. Bart Pedersen, CERN, Geneva Switzerland.

Abstract

Beam instrumentation (BI) systems at CERN require periodic verifications of both their state and condition. An instrument's condition can be diagnosed by looking for outliers in the logged data which can indicate the malfunction of a device. Presently, experts have no generic solution to observe and analyse an instrument's condition and as a result, many ad-hoc Python scripts have been developed to extract historical data from CERN's logging service. Clearly, ad-hoc developments are not desirable for medium/long term maintenance reasons and therefore a generic solution has been developed. In this paper we present the Offline Analysis Framework (OAF), used for automatic report generation based on data from the central logging service. OAF is a Java / Python based tool which allows generic analysis of any instrument's data extracted from the database. In addition to the generic analysis, advanced analysis can also be performed by providing custom Python code. This paper will explain the steps of the analysis, its scope and present the kind of reports that are generated and how instrumentation experts can benefit from them. It will subsequently demonstrate how this approach simplifies debugging, allows code re-use and optimises database and CPU resource usage.

INTRODUCTION

Both scientific and business domains have witnessed exponential growth of available data. Processing and analysing huge amounts of data is a major problem and has become it's own scientific field leading to the creation of many commercial and open source tools over the years. The need for similar tools in the Beam Instrumentation group at CERN has already been identified in the past [1]. The BI group decided to make their own tool because of the group's very specific constraints (the database can only be accessed via a dedicated java API, and users often need tailor made reports). LHC systems produce large quantities of data which can be used for checking the health of various beam instrumentation systems. OAF aims to simplify and unify the work-flow of the data analysis and problem detection.

After the evaluation of technologies on which we could base our analysis solution, Java, C++ and Python emerged as the primary candidates. These three programming languages have a long history at CERN and are widely used inside the organisation. We finally choose Python, as it offers a rich choice of scientific libraries for numerical and statistical analysis (for example we make heavy use of the Python Panda and Matplot libraries). Furthermore, prototyping in an interpreted language is also much faster than in languages requiring compilation. Python is also beginner friendly and can be used by users with limited programming experience – an important feature for our needs if OAF is to offer a *custom analysis* feature which allows broader analysis via dedicated code supplied by instrument experts.

STATUS BEFORE OAF

Data logged by the LHC is stored in a so-called logging database [2] and is accessible via a web-based user interface (Timber) which accesses data via a Java API, but provides limited analysis capabilities. This data is regularly extracted and analysed off-line by experts, to elicit useful information about an instruments performance. In the absence of a standard means to do this, instrument specialist inevitably started developing various independent tools. Each of these tools had to support the same set of operations:

- Data extraction
- Statistical analysis
- Production of a report document

The absence of any framework to guide the developers of analysis tools, lead to many problems including:

- Code duplication
- Sub-optimal means of data extraction (by performing an out of process system call, which in turn ran a generic Java command line tool)
- The data extracted by the Java command tool was subsequently dumped into a text file and then parsed into various python data structures
- Some scripts were moved to newer Python / library versions while others remained on out-dated versions
- Very often authors of scripts stayed at CERN for a limited period of time leaving the maintenance burden on newly arrived colleagues.

Dealing with this script *zoo* became a complex software engineering task in itself, and it was soon obvious that the common functionality of these various scripts had to be handled differently so that the maintenance of the infrastructure of the resulting framework should not concern the instrument experts.

DATA SOURCE

The Logging Service stores data coming from pre-defined signals into an Oracle database, and provides a Java API accompanied by a generic GUI (Timber) which can be used to extract and visualise logged data.

ISBN 978-3-95450-177-9

Forcing data retrieval only via this API allows for a better monitoring of database loads and avoids requests for the amount of data that could potentially take down the entire database. Data can be extracted by creating dedicated queries, or by the use of a so-called snapshot (a stored query identified by a unique name). Rather than using the aforementioned Java command tool, OAF retrieves data by directly calling the Java API using JPype[3] with the desired snapshot name. JPype allows the use of Java libraries directly in Python programs which is very convenient and avoids storing variables in temporary files as was done in the past. A dedicated Python module converts Java objects into python dictionaries for easy analysis by the framework.

FRAMEWORK ARCHITECTURE

One of the most basic software design principles in modern software engineering is the separation of tasks. A computer program should be divided into distinct components that, are responsible for single well defined tasks. In previous ad-hoc scripts this principle was violated and quite often a single module performed many distinct tasks (extracting data, analysis, producing the report, etc.). One of OAF's design goals was modularity which implicitly embraces the separation of tasks principle. Adopting this principle, simplifies the maintenance and development process. If for example the underlying Java API ever changes, only one module has to be updated, and the rest of the framework is shielded from this change. It also allowed different members of the OAF team to work independently on different modules after agreeing on common interfaces. The resulting OAF framework consists of following parts:

- DB connection module which is responsible for accessing and converting data to an agreed python data structure.
- Excel files which define the configuration of the analysis. Configuration is achieved by the modification of several spreadsheets. This allows the configuration of many features of the framework including a list of participants to be notified when a report has been generated, the declaration of new variables based on extracted ones, the criteria for alarms, the choice of plots, etc. It is important to note that this part is fully generic so that instrumentation experts do not have to provide any code in order to configure this part of the analysis.
- Report Generator GUI. OAF is a command line tool with parameters such as the date of the analysis, snapshot name etc. provided as command line arguments. The Report Generator is a GUI which simplifies the request for a new report, with for example, dates chosen from a calendar widget.
- A module with generic analysis code. The common part of all analysis is performed in this module with its configuration retrieved from the Excel spreadsheet.
- Optional modules with custom code. In the case where more sophisticated analysis is required, OAF provides a place holder for dedicated code to be provided by the instrumentation expert.
- Web application which allows browsing existing reports in an easy and user friendly way.

ANALYSIS FLOW

Automation is an important aspect of the OAF framework. A typical analysis can be divided into separate phases, which are well defined and roughly correspond to the functionality of the modules described previously (Fig. 1).

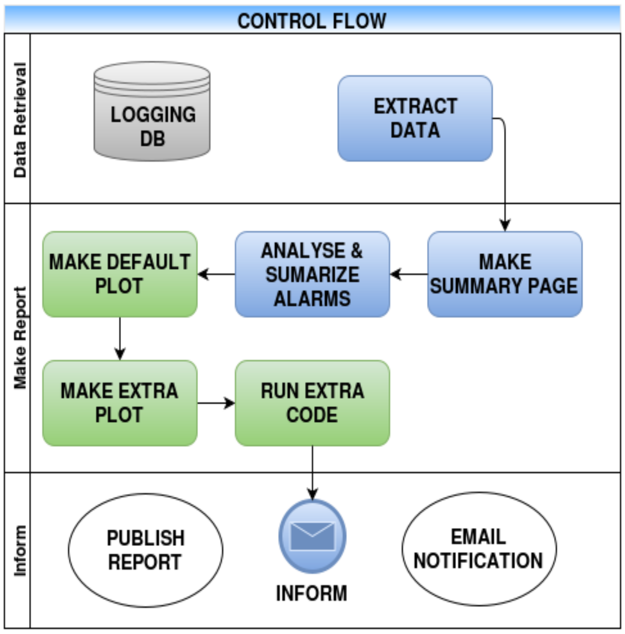

Figure 1: Analysis control flow. Blue: mandatory steps. Green optional steps.

Analysis is fully automated and can be invoked on a daily basis by a time-based scheduler (Linux CRON).

OAF provides generic core functionality to perform each of the following steps during an analysis:

- Extract Data: retrieve data from the database and perform data conversions
- Make summary page: describe the scope of the report, along with devices and variables used
- Analyse alarms: provide statistics on extracted data and perform checks with user-configured constraints
- Make default plots (Optional): produce results and reports of available data and alarms
- Make extra plots (Optional): produce more complex plots such as FFTs etc
- Run Expert Code (Optional): runs third party code submitted by the expert
- Inform: Depending on provided configuration E.g. sends emails with the report, stores report for further analysis

STRUCTURE OF THE REPORT FILE

The first page of the report provides information about devices and variables covered by the analysis and specified time window, with a summary of any alarm raised. The rest of the report is made up of plots and tables as defined by the configuration provided in the Excel spreadsheet. For each device retrieved, plots of the extracted data along with any related alarms are generated.

For example, the BLM-LHC-temp report diagnoses the condition of an LHC beam loss monitor acquisition card. In this case the temperature of the card is the parameter we want to monitor. Figure 2 shows this measured temperature for 16 different cards for a given VME crate. Each point on the graph contains statistics such as average, minimum and maximum values along with the standard deviation for a daily measurement. The green band indicates the range of allowed temperature values. In figure 2 we can immediately identify six outliers indicating a failure of temperature measurement for cards 5 to 7 and cards 13 to 15.

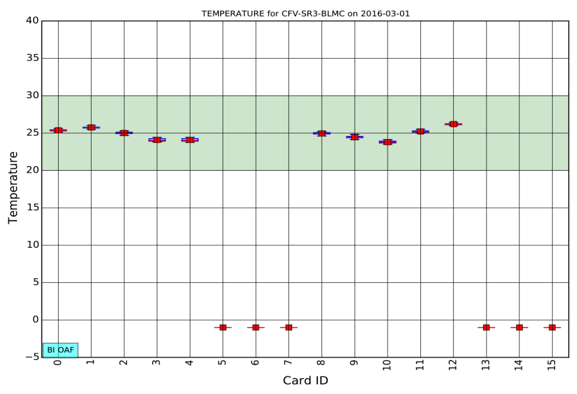

Figure 2: Example of an OAF plot.

There are numerous alarms, various plot types, and several facilities to perform variable conversions which relieves the user from writing such code. OAF can raise alarm in the following instances:

- DISCRETE: alarm raised if values do not belong to the given list of values.
- MEASURE: alarm raised if values do not remain within given range.
- STATUS_BIT: alarm raised if the state of status bit register does not correspond to the nominal state of each bit
- SWITCH: alarm is raised each time a value is changed.

Sets of available plots includes:

- HISTOGRAM
- FFT
- STATISTICS (mean, deviations)
- and many others..

Quite often we need to transform the extracted data in order to carry out further analysis. The most common conversion types have been identified and are made available to the user. Generated this way variables can be plotted in the generic plots, as well as being used for the alarm checking. More than twenty types of conversions are available including statistics calculations, FFTs, etc.

EXAMPLE USE CASE

The daily BCT_SPS_safety reports, provide offline monitoring of the 2 DC current transformers used in the SPS North Area for personal safety matters. The report contains a check which compares the recordings of these 2 DCCTs during the last 24 hours. All differences are reported as a potential error of one DCCT. As an example we can see in Fig. 3 that a spurious signal appeared on BCT_897 (blue trace) around 00:45 on the 7th of August 2016. This automatic analysis is particularly useful to track this kind of rare event.

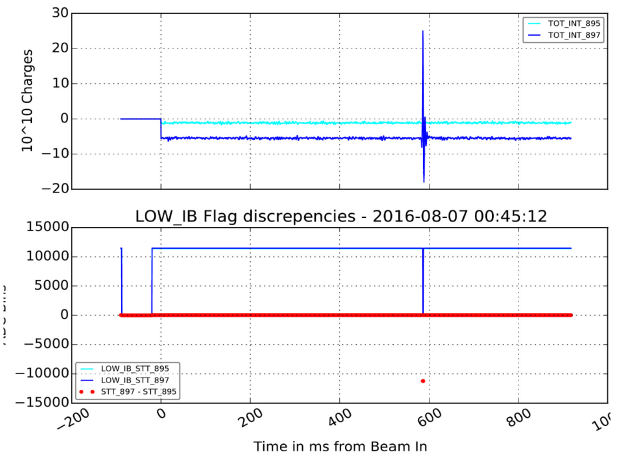

Figure 3: Top Plot : Signal proportional to DC beam current during an SPS Cycle (one trace for each DCCT). Bottom plot : Shows the evolution of the 2 status flags during the cycle (high level = OK for extraction, low level = NOT OK for extraction) and in red, the difference of the two flags.

RESULTS

OAF analysis tasks are executed every night by an automatic CRON task. After the analysis is performed, interested recipients receive an email with a short summary of the findings (how many alarms ware raised etc.) and a link to the report. Adding a configuration of new analysis is done via an Excel configuration file, which is a much more robust and easier than the previous approach (creating dedicated python script). We can easily extend OAF to support more types of alarms or plots just by updating the generic part of the framework without breaking backward compatibility with existing analysis tasks. Making such changes previously required separate modification of each python script.

Presently, some forty reports are produced every day, covering beam position measurement, loss

ISBN 978-3-95450-177-9

measurement, current measurement and profile measurement in all of CERN's accelerator complex (LHC, SPS, PS, PS BOOSTER...). Two thirds of these reports only rely on the generic code and features. Some reports use expert code included into the framework to add some specific analysis and plots. Finally, a recurrent “OAF outcomes” topic has been added to our regular internal technical board meetings where we present interesting observations to all BI experts as well as explaining any new features of the framework.

ACKNOWLEDGEMENT

We thank P. Odier, CERN, for providing information on BCT_SPS_safety report usage.

REFERENCES

[1] S. Jackson, “A framework for off-line verification of beam instrumentation system at CERN“, in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’13)*, San Francisco, USA, Oct. 2013, paper MOPPC139, pp. 435-438

[2] C. Roderick et al., “The LHC Logging Service: Handling Terabytes of On-line Data”, in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2009)*, Kobe, Japan, Oct. 2009, paper WEP005, pp. 414-416.

[3] `http://jpype.sourceforge.net/`