DAΦNE TECHNICAL NOTE
INFN - LNF, Accelerator Division

Frascati, 07/02/2014
Note: **C-21**

DJANGO: A TOOL FOR THE ANALYSIS AND PRESENTATION OF THE DAΦNE BEAM INTEGRATED CHARGE DATA

F. Agostini, P. Ciuffetti, A. Stecchi

Introduction

The DAΦNE operation involves the transport of electrons/positrons - through the transfer lines - from the LINAC up to the Main Rings. The knowledge of the injected charge integral into the accumulator and the main rings over time is very important, both for a systematic evaluation of the beam transport efficiency and for matters of safety.

In this paper we describe an application dedicated to the post-processing, storage and presentation of the data produced by the DAΦNE Charge Transport Monitor System (CTMS). This application integrates different software technologies such as: MySQL database, Java, Javascript and PHP.

The objectives of the DJANGO (Data analysis with JAva and Nifty Graphical Output) project are:

- to process the data coming from the CTMS log files and push them into a relational database;
- to present the acquired data through a user-friendly WEB application.

1. Data Description

During the DAΦNE operation the e+/e- beam is driven to different machine zones depending on the specific machine phase:

- LSP LINAC to Spectrometer;
- LBT LINAC to BTF;
- LTA LINAC to Accumulator;
- AMR Accumulator to Main Ring.

In these phases, the beam runs different paths and crosses different sequences of the beam charge monitors installed along the transfer lines (see Table 1).

Table 1: Charge monitors involved in the different machine phases both for e- and e+ modes.

Machine phase	Electrons mode	Positron mode
LINAC to Spectrometer	WCMTM001	WCMTM001
LINAC to BTF	WCMTM001 WCMTB002	WCMTM001 WCMTB002
LINAC to Accumulator	WCMTM001 WCMTT001 WCMTL001	WCMTM001 WCMTT001 WCMTR001
Accumulator to Main Ring	WCMTE002 WCMTT001 WCMTR001 WCMTT002 WCMTE001	WCMTT001 WCMTL001 WCMTT002 WCMTP001

The CTMS performs a real-time acquisition of the signals coming out from the charge monitors. It continuously stores the last acquired charge values into the central live storage area (which is used for live presentations), increments the total charge values in temporary histogram memories and periodically dumps these histograms to log files.

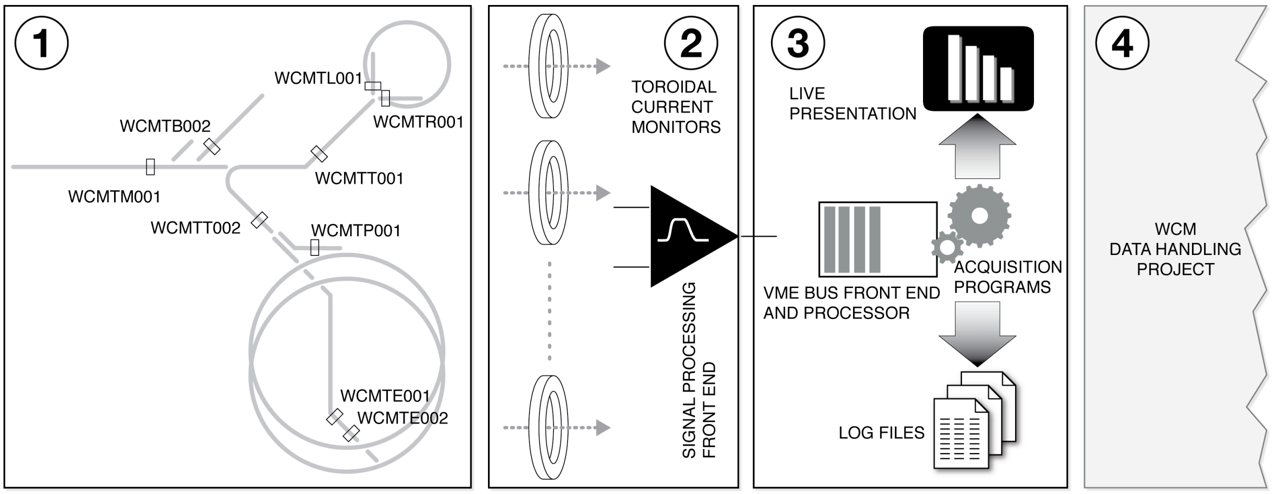


Figure 1: Schematic representation of the already existing CTMS.

1) disposition of the toroidal current monitors along the DAFNE transfer lines complex;
2) the signals coming from the toroidal current monitors are integrated by a dedicated front-end which returns a voltage level proportional to the beam charge;
3) a program - running on an embedded VME processor - acquires the signals by mean of a triggered ADC, updates the data into the Control System live storage area and produces log files for further analysis.

2. Project Workflow

The CTMS retains the beam charge acquired data into the RAM of the VME processor (see Fig. 1, frame 3). These data are consequently volatile and will be lost in case of a CPU reboot (e.g. for modification to the software or hardware maintenance).
For such reason, the VME processor dumps the content of its histogram memories to log files every minute and - should the processor reboot - the histograms written to the log files will restart from zero.

This means that the log files are not suitable for a direct read-out of the total charge integral but must be post-processed to resolve the eventualities described above. This has led to the DJANGO project.

DJANGO accesses the log files (following the creation date order) and calculates the incremental sum for each charge monitor. The resulting data (monotonically increasing) are then stored into a SQL database to ease the correlation of the monitor sequences with the different machine operational phases (see Tab. 1). The graphical presentation is performed by mean of a WEB application that accesses the database through SQL queries and plots the data using an up to date JavaScript library. The entire workflow of the DJANGO architecture is shown in Figure 2.

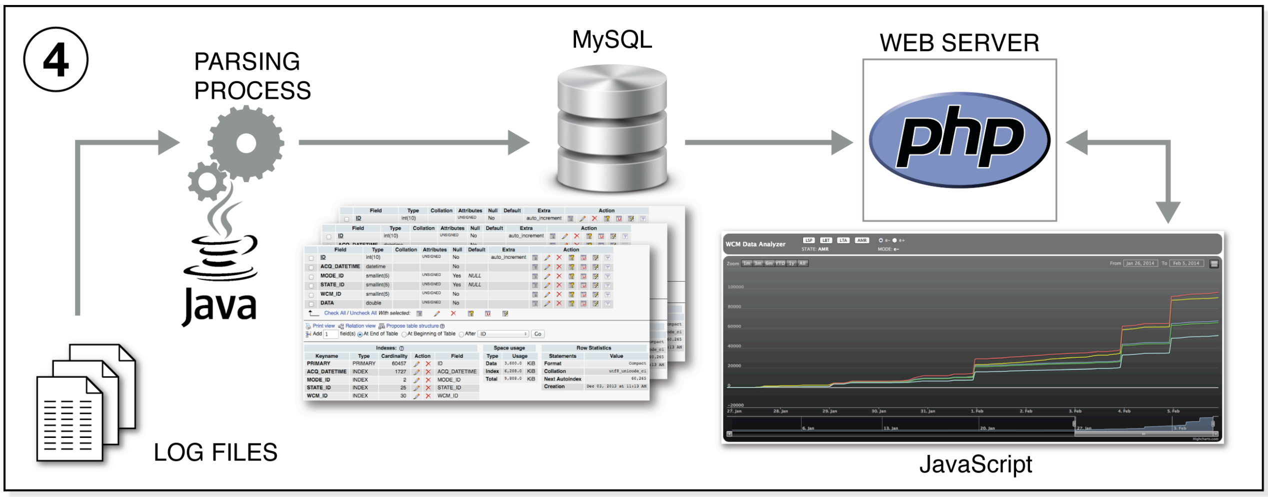
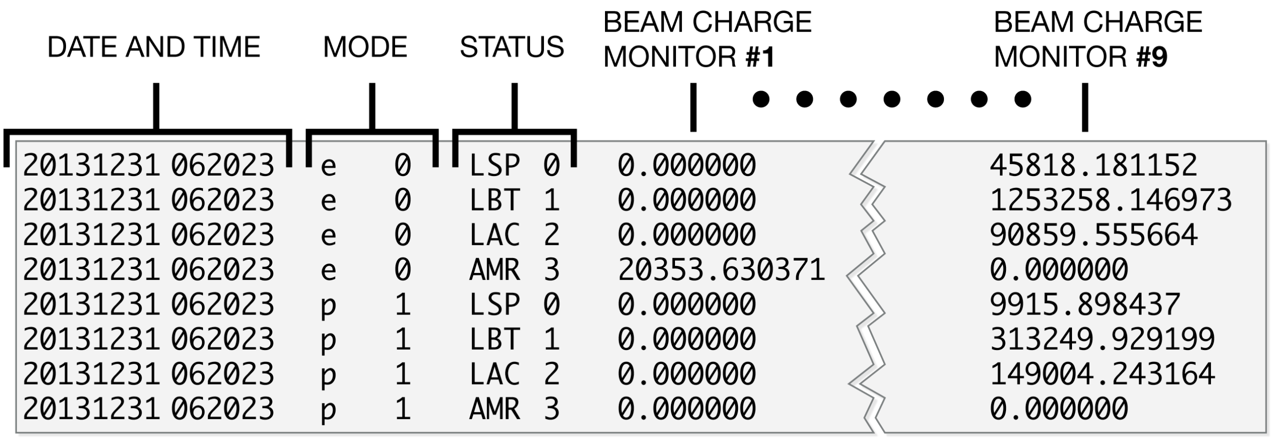


Figure 2: Workflow of the DJANGO project. A Java application processes the log files produced by the CTMS and pushes the elaborated data into a MySQL database. A WEB application - employing JavaScript on the client side and PHP on the server side - displays an interactive plot of the integrated charge for the different machine states and modes.

3. Java Application

The purpose of the Java application is to parse all the log files produced by the CTMS, which have not been parsed yet in some previous application runs. The application finds out and resolves all the "restart-from-zero" occurrences, calculates the integral value for every current monitor for any mode/status and streams the processed *recordset* to the database with a time resolution of one hour.


DATE AND TIME
MODE
STATUS
BEAM CHARGE MONITOR #1
BEAM CHARGE MONITOR #9
• • • • • • •

```
20131231 062023  e   0   LSP  0   0.000000           45818.181152
20131231 062023  e   0   LBT  1   0.000000           1253258.146973
20131231 062023  e   0   LAC  2   0.000000           90859.555664
20131231 062023  e   0   AMR  3   20353.630371       0.000000
20131231 062023  p   1   LSP  0   0.000000           9915.898437
20131231 062023  p   1   LBT  1   0.000000           313249.929199
20131231 062023  p   1   LAC  2   0.000000           149004.243164
20131231 062023  p   1   AMR  3   0.000000           0.000000
```

Figure 3: Example of a log file produced by the CTMS. The figure shows one of the *recordset* that are appended every minute to the file (the ellipses indicate the series of values for the intermediate monitors).

The application performs also a consistency check, storing into the database only if all the data for a given day is present. Consequently the log file created the same day in which the application is running is ignored, because it is still being filled.

The Java application has been developed following an MVC (Model, View, Controller) architectural pattern, so to separate internal representations of information from the way the information is presented to or accepted from the user (in our case the View part of the pattern is done by the JavaScript).

A brief description of the Java classes follows (see Fig. 4):

- FileParser: this class is used to parse every log file, excluding those already processed and pushed into the database. The log file created the same day the software is ran on is also ignored to avoid partial parsing;
- ListFiles: this class is used to obtain a list of all the log files present in the log directory, excluding those that are not WCM log files;
- SyncDb: this class contains the main method for the Java application. It is used to parse all the log files and make them ready for the database;
- MySQLConnection: this class is in charge of connecting, reading and writing from/to the database;
- SamplingData: this is the Entity class that represents a log file row. It is used during the parsing to identify row-by-row data;
- LastWcmData: this is the Entity class that represents the last data stored into the database.

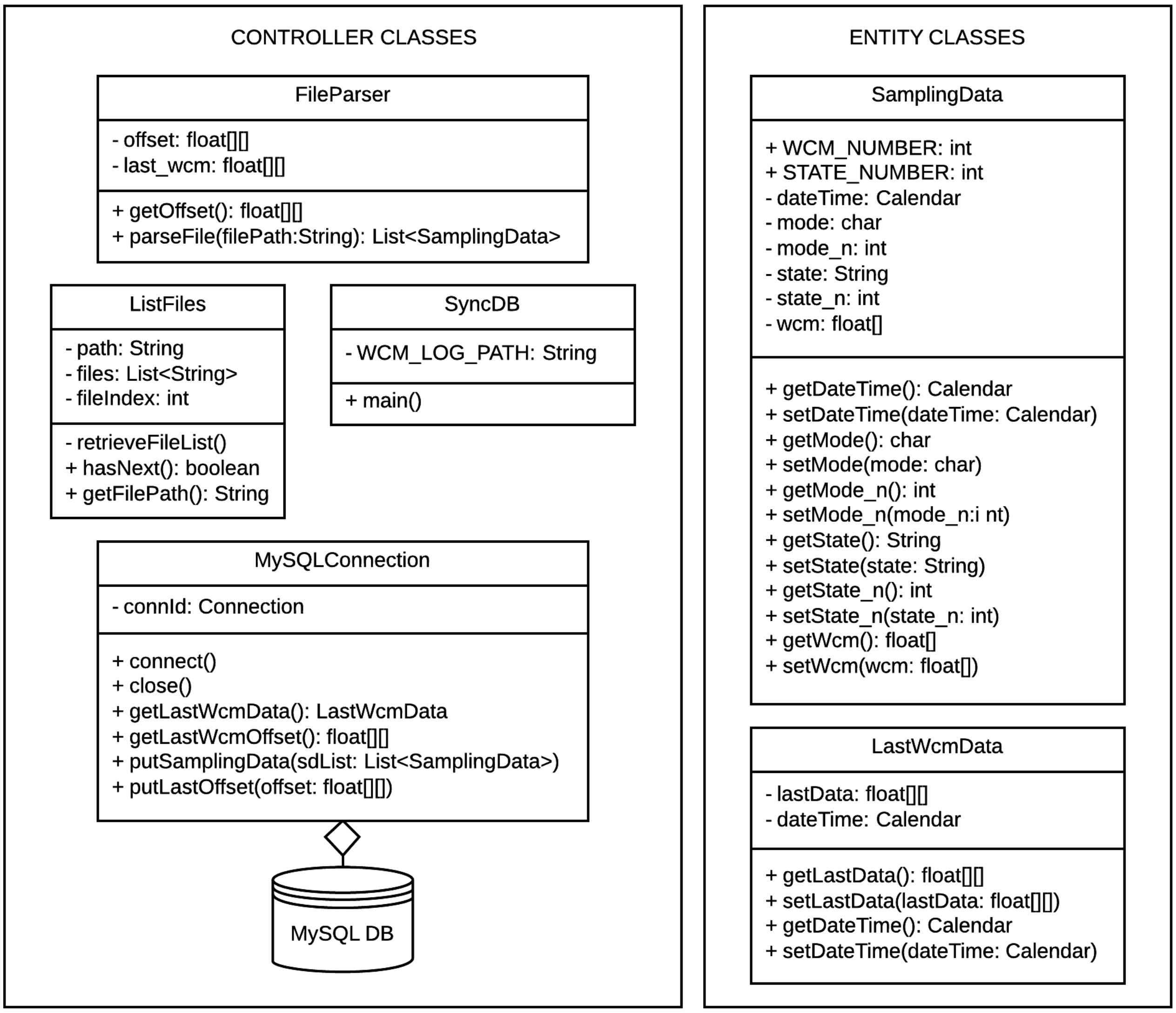


Figure 4: Class diagram.

4. Database Structure

The structure of the MySQL database consist of 5 tables (see Fig. 5):

- WCM: This table contains all the WCMs with their position in the machine and their state (online, offline);
- MODE: This table contains all the possible mode in which the machine can run (electrons, positrons);
- STATE: This table contains all the possible states in which the machine can be while running:

▪ LSP	LINAC to Spectrometer;
▪ LBT	LINAC to BTF;
▪ LTA	LINAC to Accumulator;
▪ AMR	Accumulator to Main Ring.

- ACQUISITION: This table contains all the raw acquisitions from all the beam charge monitors;
- ACQ_OFFSET: This table contains the last raw values acquired from the last parsed file, witch are used from the parser to properly calculate the integral across parsing actions.

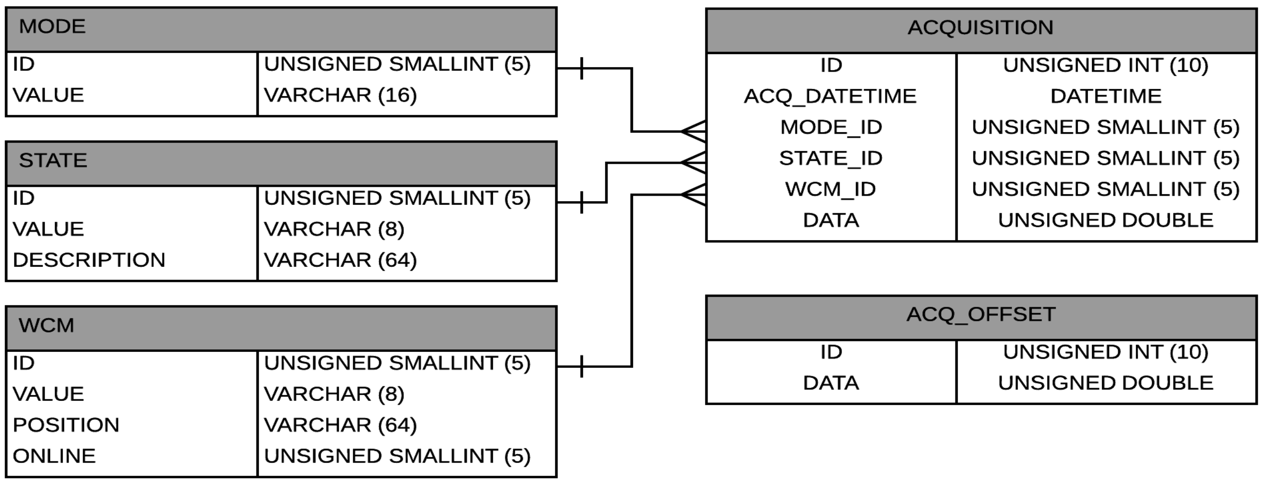


Figure 5: Database ER Diagram.

The MODE table:

ID	0	1
VALUE	e	p

("e" and "p" represent the electron and positron modes)

The STATE table:

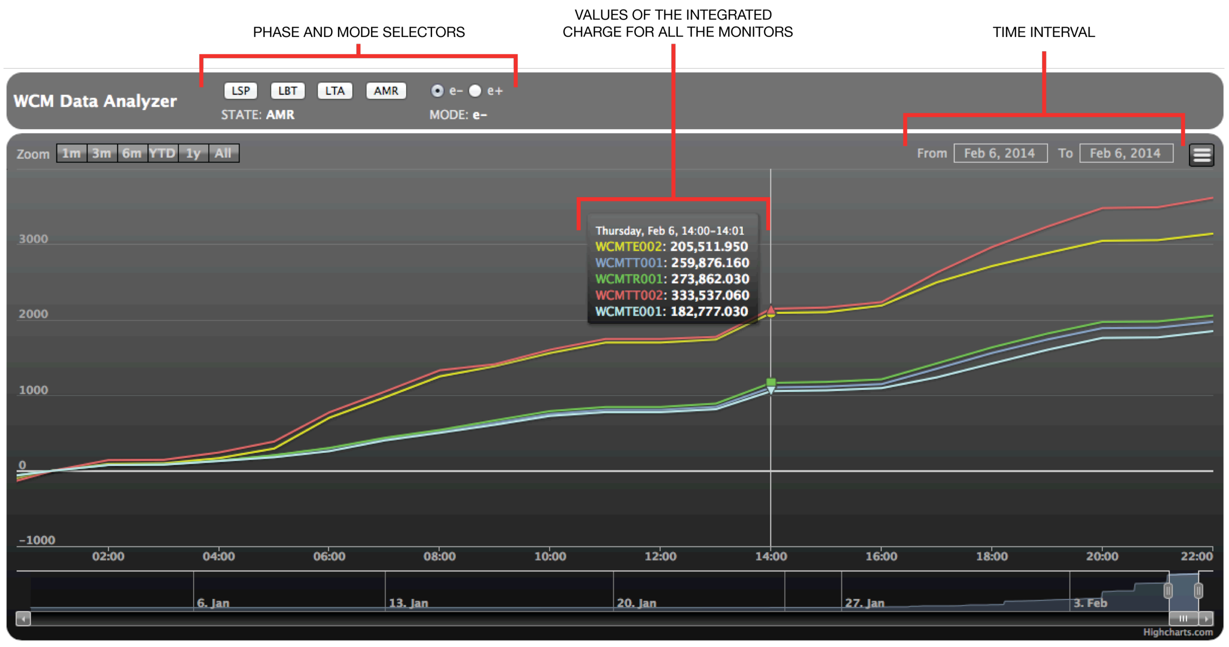
ID	0	1
VALUE	LSP	LBT
DESCRIPTION	LINAC to Spectrometer	LINAC to BTF
ID	2	3
VALUE	LTA	AMR
DESCRIPTION	LINAC to Accumulator	Accumulator to Main Ring

The WCM table:

ID	0	1	2	3	4
VALUE	WCMTE001	WCMTP001	WCMTT002	WCMTL001	WCMTR001
POSITION	…	…	…	…	…
ONLINE	1	1	1	1	1
ID	5	6	7	8	
VALUE	WCMTT001	WCMTB002	WCMTE002	WCMTM001	
POSITION	…	…	…	…	
ONLINE	1	1	1	1	

5. Graphical Presentation

In order to visualize the data coming from the MySQL database we have adopted *Highstock*[1]. *Highstock* is a pure JavaScript charting library - working in any WEB browser - that allows you to create general timeline charts. It includes sophisticated tools such as selectors for date and time and controls for zooming and panning. Among the many *Highstock* available charts, we have chosen the *Compare multiple series* plot, which is very suitable for plotting the integral charge for many monitors at a whole. Finally, with the *exporting module* enabled, it is also possible to save the charts as graphic files in PNG, JPG, PDF or SVG formats and print them directly from within the WEB page (see Fig. 6). The data transfer between MySQL and the *Highstock* application is obtained by mean of a PHP script that generates JSON (JavaScript Object Notation) data for any given request coming from the *Highstock* application. The PHP script accesses the MySQL database in read-only mode, to ensure data security and avoid SQL injection.



Thursday, Feb 6, 14:00-14:01
WCMTE002: 205,511.950
WCMTT001: 259,876.160
WCMTR001: 273,862.030
WCMTT002: 333,537.060
WCMTE001: 182,777.030

Figure 6: *Highstock* data visualization through a WEB browser.
The four press buttons and the two radio in the top bar of the windows allow to select the wanted machine phase and the electrons or positrons mode. After any action on these controls, the chart dynamically updates showing the integrated charge for all (and only) the monitors involved in the selected phase and mode (the vertical scale is in Arbitrary Units).
The markers at the bottom allow zooming in/out the time span and the scroll bar to pan over time.

6. Performances

In our test the Java parser has run on a server Sun Fire v20z with an AMD Opteron @1.8GHz, 2GB RAM, with Linux CentOS 5.5 x86_64 and Java SE Runtime Environment 1.7.0_51.
On this setup it takes 0.75 s to the Java parser to process a single daily log file. This time is obviously required only once because since data are put in the database, the log files are no longer used.

The drawing time depends on the length of the period and the number of monitors that have to be shown for the requested status and mode.

The drawing time for 1 year of data is 3.2 s/$N_{m,}$ where N_m is the number of monitors.

References

[1] Highsoft AS: http://highsoft.com