

CMS Internal Note

The content of this note is intended for CMS internal use and distribution only

September 12, 2003

Data Validation Software Architecture during the CMS ECAL Test Beam

Giovanni Organtini

Università di Roma "La Sapienza" & INFN-Sez. di Roma, Roma, Italy

Alessandro Di Biagio, Massimiliano Maggiori

Università di Roma "La Sapienza", Roma, Italy

Abstract

This note describes the general architecture envisaged for the Data Validation tasks during the CMS electromagnetic calorimeter test beam to be taken in August 2003. The proposed architecture allows the parallel development of language independent data validation programs without the needs of synchronization and complex user interface. The only requirement for the data validation subtasks is that their output must be available as HTML-formatted files.

1 Introduction

During the summer of 2003 a test beam will take place to prepare and test software to be used for the precalibration of CMS [1] electromagnetic calorimeter supermodules in 2004. The construction schedule is rather tight. As a consequence, only few calorimeter supermodules will be exposed on electron beams. The rest have to be calibrated using physics events.

The available time in 2004 for precalibration is limited and the efficiency of the precalibration is then very important, since there will not be a second chance. During the next test beams, raw events will be automatically processed just after data acquisition. The relevant physics quantities for calibration will be available few minutes after the end of each run. These data have to be validated online providing a prompt identification of potential problems and, in this case, people on shift and run coordinators have to be notified as soon as possible to take the appropriate actions.

In this note we will describe the proposed architecture for the general framework of the data validation tasks.

2 Data Validation Requirements

Single data validation tasks can be defined as jobs with a well defined objective that gets reconstructed events as well as raw events as input, analyze them and produce some output that can be useful to shifters and to coordinators to define the quality of the data. Examples of these tasks are:

1. analysis of the shape and parameters of the deposited energy spectra;
2. analysis of stability of the pedestals;
3. evaluation of a first order calibration constant;
4. analysis of the noise in electronic channels;
5. check of the quality of thermal sensors data;

and so on. Each task is relatively simple and fast. The number and the quality of the tasks can vary even during the test beam period and the general framework must take it into account. The output of each task can be composed of text and images useful to detect potential problems or to show the preliminary results obtained on the fly. Each task can even detect problems by itself (for example, if the shape of a distribution is far from the expected one) and in this case the framework must allow the task to notify it to responsible people as early as possible and in an effective way.

Shortly, the requirements for the data validation framework are:

1. to be able to automatically launch any number of data validation tasks, even if this number varies with time;
2. to be able to manage data validation tasks written in any possible language;
3. to make data validation tasks development independent on the details of the framework;
4. to provide easy access to information produced by data validation tasks in real time;
5. to automatically manage alarms and notifications produced by data validation tasks.

All these requirements must be fulfilled by the framework without the need of its software modification or recompilation.

To achieve this goal we developed a Java application based on the so called Mediator-Observer Pattern [2], able to detect Events from Sensors and to dispatch them to Solutions.

We will describe the framework details in the next section.

3 The Mediator-Observer framework

The UML [3] model of the framework can be seen in Fig. 1. The most important classes in the model are the Mediator, the Solution and the Sensors.

Sensors and Solutions are Colleagues referring to the Mediator to exchange messages between them. The Mediator collects Events detected by Sensors and notifies them to Solutions. To this respect, a Solution is an Observer and the Mediator is a Subject.

Sensors are objects able to detect events (the end of a run, a change in the content of a given file, the saturation of a threshold, etc.). Each time a Sensor detects an event it builds an Event object containing a description of the detected event and the identity of the Sensor. The Event is then sent to the Mediator, who, in turns notifies it to the whole collection of registered Solutions.

Solutions examine the Event and, based on the Sensor identity and its content, may take the decision to make an action or to discard it. More than one Solution can handle a given Event.

This design decouples completely the detection of an Event from its handling and leaves the responsibility of discriminating events only to the Solution objects. With this design it is possible to develop several independent Solutions for the same event and simplifying a lot the development of sensors who must not take into account the possible receivers. Moreover, the only object that knows how many sensors and how many solutions exist is the Mediator and this makes it possible to add sensors or solutions later without the need of complex software manipulation.

The design illustrated above has been implemented in Java and few concrete Sensors and Solutions have been implemented to take into account the needs of the data validation at the ECAL test beam.

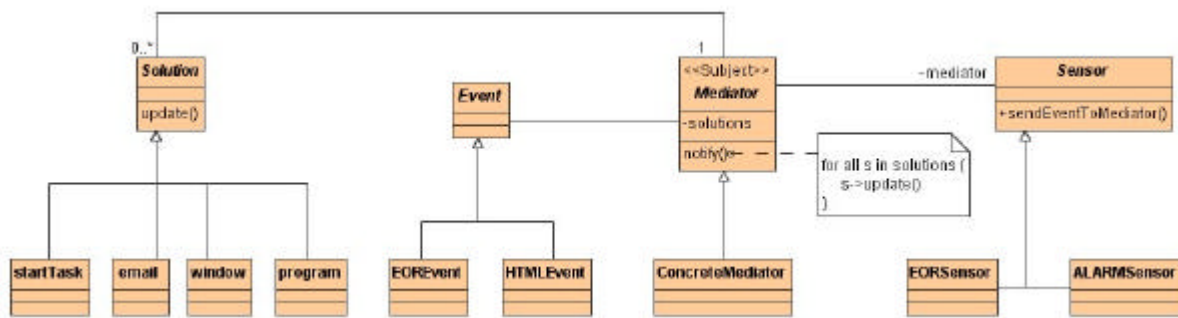


Figure 1: The UML class diagram of the framework

4 Test beam Sensors and Solutions

Two Sensors are envisaged for the operations at the test beam:

1. an End Of Run (EOR) Sensor able to detect the end of a run (here a run is defined as an uninterrupted data acquisition period whose data are collected in one single file). In fact this Sensor must be able to give an Event whenever a new pre-processed data file is available for the data validation tasks. The Event will contain the run details such as the name of the produced file and all other parameters that may be needed, according to what the content of it will be.
2. a Generic Alarm (ALARM) Sensor able to detect alarms and warnings issued by data validation tasks.

The number of available Solutions is four:

1. a Start Data Validation Task (startTask) Solution. This Solution intercepts the EOR Event issued by the corresponding Sensor and must be able to load and starts any number of data validation tasks (i.e. programs or scripts) to analyze the last pre-processed run. The way in which this is done must be dynamic, i.e. no software modification nor recompilation is needed for the Solution if data validation tasks are changed, added or removed.
2. a Send A Mail (sendMail) Solution. This Solution intercepts the ALARM Event and will send an e-mail to a list of addresses with a given text: both the mailing list and the mail body will be part of the Event content. This Solution is activated only if the Event content requires it and is used to notify the relevant people in order to let them to take the appropriate actions, if any.

3. a Notify Message (Message) Solution. This Solution intercepts the ALARM Event too and is activated if the Event content specifies that a given message must be notified to shifters. In this case the Solution will show a popup window showing the message to the shifter. The window will have an acknowledge button that will make it disappear and its content can be either text and/or graphics.
4. a Take Action (Action) Solution that will automatically launch a given procedure (a program or a script) identified by the Event content and that is triggered by the ALARM Sensor. It could be useful to run a well known procedure in case of known problems without human intervention.

We believe that the above mentioned Sensors and Solutions will be enough to cover all the possible aspects of the data validation tasks.

The number of data validation tasks and the possible automatic procedures for prompt recovery of known problems is left open since these are automatically started by the startTask Solution and the Action Solution, respectively, provided they are copied into given directories, inspected by the Solutions to determine the tasks to be started.

In the following Sensors and Solutions will be described in detail.

5 Sensors

Both Sensors parse files to detect changes within them.

At the end of each run, in fact, a new file is available on the data disk containing preprocessed data. A daemon lists the content of the directories containing those files and, if new files appear, it appends a new line to a file called `runlist` formatted as follows:

```
<dirname> runno [parameters]
```

where `dirname` is the name of the directory containing the data validation tasks to be started, `runno` the run number and `parameters` are values whose number, type and values may depend on `dirname`. Several types of runs, in fact, are expected to take place: beam, pedestal, laser, etc. `dirname` is set according to the run type so that dedicated data validation tasks can be started properly. The EOR Sensor just parses the `runlist` file and creates an Event once a new line appear at its bottom.

Data Validation Tasks are fast and dedicated programs that compute values from input files containing the data collected during the runs and produces HTML files as output. These files can be viewed by shifters and coordinators by means of any Internet browser even remotely. In this way output files can contain both text and graphics and its rendering is left to existing products. The list of all the possible output files is automatically created on the fly by a CGI Perl script that lists the directory where they are supposed to be and produces a list of links: clicking on one of them will show the corresponding file. The text shown in the link is obtained from the title of the HTML page. In this way new Data Validation Tasks can be added at any time without the need to update HTML pages.

If a Data Validation Task detects a potential problem in the analyzed data, it adds an HTML comment to its output file. An HTML comment is a text embedded within special tags: `<!--` and `-->`. HTML comments are not rendered by Internet browsers; however they are used by ALARM Sensors to detect potential problems. Once started, in fact, the Mediator object creates as many ALARM Sensor instances as many output files. Each Sensor parses a single output file. If an HTML comment appears within the file, starting with the string `SensorHTML` (to distinguish it from ordinary comments), the corresponding Sensor will generate an Event whose type and content depends on the comment itself.

There are currently three possible Event type: e-mail, window and program. Each type of Event is used by a corresponding Solution, respectively, to:

- 1) send an e-mail to a list of people notifying them the occurrence of a problem with an arbitrary text;
- 2) open a window on the screen of the shifter notifying him/her the occurrence of the problem with an arbitrary text and eventually including graphics;
- 3) launch a program or a script to automatically fix the problem, if possible.

In order to activate one or more of these activities the HTML comment must be formatted in the proper way. To create an Event that will activate a Solution to send e-mails (remember that Sensors only generate Events, while Solutions are devoted to interpret them and act consequently), the HTML comment content must be formatted as follows:

```

emailFlag=true
<emailBody>
senderEmail={e-mail address of the user to which the data validation task
            belongs}
senderName={data validation task name (or its author name)}
receiverEmail={comma separated list of recipients}
smtpServer={the SMTP server name - usually localhost}
subject={the subject of the e-mail, usually a brief report of the problem}
<message>
{any text shown in the message body}
</message>
</emailBody>

```

Curly brackets are used to show and describe variables. If the Sensor detects the `emailFlag=true` string within the comment it generates an Event including the content of the `emailBody` tag.

To generate a window Event, the comment must be formatted as follows:

```

winFlag=true
<winBody>
{any HTML formatted text}
</winBody>

```

while to generate a program Event, the required format of the comment is:

```

commandFlag=true
<commandBody>
{any text representing a file path}
</commandBody>

```

6 Solutions

Solutions are notified by the Mediator as soon as an Event has been generated by a Sensor. For each Event type one or more Solutions can be activated. Presently the possible Solutions are four.

The `startTask` Solution activates whenever an EOR Event has been generated. The Solution takes the run type, the run number and all other needed parameters from the Event. Depending on the run type, the Solution lists all the executable files existing in the corresponding directory and launches them in separate threads. In this way an arbitrary number of Data Validation Tasks can be launched after each EOR without the need of modify the source code of the Framework, nor to update configuration files. In order to add a new Data Validation Task to the list of the ones that should be executed, it's enough to copy its executable in a given directory. Moreover, since the tasks are launched as separate, standalone programs, there are no requirements on the language to be used to write them and no dedicated library should be included in them. In this way the tasks can be developed by different people without the need of strong coordination and almost ignoring the details of the Framework.

The `sendMail` Solution is activated after an e-mail Event has been notified to it. The Solution takes the e-mail parameters from the Event body and initiates a dialog with the specified SMTP server to send the message embodied within the Event to the recipient list.

If a window Event is intercepted by the Message Solution, it launches an Internet browser rendering the HTML code contained within the Event body. In order to be flexible enough, in fact, the HTML code is first copied onto a file, and then a dedicated shell script is launched with the file name as parameter. The script set up needed variables (for example the `DISPLAY` environmental variable if the message should be shown on a different screen) and really launches the preferred browser. In this way the user is completely free to choose and customize its own Internet browser modifying the shell script, rather than the Solution Java source code. In our application we used a specially configured Mozilla browser in order to show plain windows without toolbars and input boxes.

Finally, the Action Solution activates if a program Event has been generated. This Solution just takes the program path from the Event and executes it in a separate thread. Again the program can be written in any

language and no libraries must be linked with it to work within the Framework.

Of course, more than one Solution can be activated at the same time, depending on the generated Events.

7 Conclusion

We have realized a very flexible framework for online data validation. It allows experts to write data validation tasks completely independently on each other, on the framework details and on external libraries. The only requirement being the need to have HTML formatted files as output. This choice has also the advantage of providing a unique, simple and friendly tool to navigate within all the output files: in fact we provided a CGI Perl script that will automatically produce a list of links on the fly, showing all the available output files to shifters and coordinators, even remotely.

With our design the number and the type of data validation tasks can be arbitrary and there is no need for source code modifications or configuration update.

The design is based on the Mediator-Observer patterns, in which independent Sensors generate Events and send them to the Mediator who, in turns, notifies all the Solutions (Observer). The latter activate themselves only if they recognize the Event, otherwise they ignore it.

Due to the great flexibility of the framework we expect it to be usable in many other fields.

References

- [1] The Compact Muon Solenoid, Technical Proposal, CERN/LHCC 94-38 (1994)
- [2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: elements of reusable object-oriented software", Addison-Wesley
- [3] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley