The Offline Framework of the Pierre Auger Observatory

Lukas Nellen*, I de Ciencias Nucleares, UNAM, 04510 Mexico, D.F.
Stefano Argiro, University of Torino, Italy
Thomas Paul, Northeastern University, Boston
Troy Porter, Louisiana State University, Baton Rouge
Luis Prado Jr, State University of Campinas, Campinas

Abstract

The Pierre Auger Observatory [1] is designed to unveil the nature and the origin of the highest energy cosmic rays. Two sites, one currently under construction in Argentina, and another pending in the Northern hemisphere, will observe extensive air showers using a hybrid detector comprising a ground array of 1600 water Cherenkov tanks overlooked by four atmospheric fluorescence detectors. Though the computing demands of the experiment are less severe than those of traditional high energy physics experiments in terms of data volume and detector complexity, the large geographically dispersed collaboration and the heterogeneous set of simulation and reconstruction requirements confronts the offline software with some special challenges.

We have designed and implemented a framework to allow collaborators to contribute algorithms and sequencing instructions to build up the variety of applications they require. The framework includes machinery to manage these user codes, to organise the abundance of user-contributed configuration files, to facilitate multi-format file handling, and to provide access to event and time-dependent detector information which can reside in various data sources. A number of utilities are also provided, including a novel geometry package which allows manipulation of abstract geometrical objects independent of coordinate system choice. The framework is implemented in C++, follows an object oriented paradigm, and takes advantage of some of the more widespread tools that the open source community offers, while keeping the user-side simple enough for C++ non-experts to learn in a reasonable time. The distribution system includes unit and acceptance testing in order to support rapid development of both the core framework and contributed user code. Great attention has been paid to the ease of installation.

INTRODUCTION

The offline framework of the Pierre Auger Observatory is the central backbone of all simulation, reconstruction, and analysis work done by the collaboration. As such, it has to meet the following requirements:

- Flexibility to accommodate different types of analysis
- Ease of use for the physicist
- Ease of installation
- Maintainability over the experiment lifetime of 20 years and for some time beyond.

To satisfy the requirements, the offline framework is implemented

- in C++
- in a highly modularised fashion
- relying wherever possible on well-supported, open standards like XML,
- and avoid locking into a single provider solution.

The framework consists of several major sub-packages containing several components. The contained packages are Utilities, Framework, EventIO, and Modules. Additionally, the offline core provides database utilities for processing and preparing the data that are to be made available via the database interfaces in the offline framework. In Modules, we include both core modules that provide administrative and support functions and modules that contain physics code. The later are expected to be distributed independently in the future. The dependence within the sub-packages is strictly non-circular, following the order above.

UTILITIES

The Utilities sub-package contains services that are, in general, not specific to the needs of the Pierre Auger Observatory. The utilities include: specialised C++ service libraries, error logging, mathematics and physics services, a geometry package [2], configuration parsing, and testing.

EVENT IO

The event IO libraries provide access to different formats of event storage. The native event format can store the full information of the offline event, while other formats, like the raw data acquisition formats [3, 4] hold only part of the information available in the event. Some formats yet, like the output format of air shower simulation programs [5, 6], are read only.

The event IO libraries can be called by any piece of code using the framework. Particularly, the core framework provides a simple set of modules for reading and writing events. The real work is deferred to the Event IO libraries.

Combining the native IO format with simple reader and writer modules, it is possible to dump the event at any given processing stage and resume processing later, in a separate

* lukas@nuclecu.unam.mx

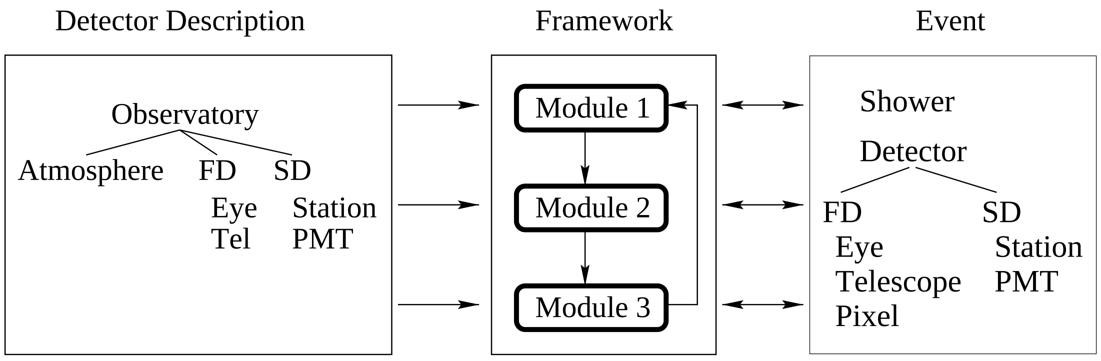


Figure 1: General structure of the offline framework

job. This feature is important for the creation of event subsets or to avoid the repetition of computationally intense steps in an analysis-reconstruction chain. Libraries of partially simulated or reconstructed events can be created this way.

FRAMEWORK

The framework of the offline software consists of three main parts:

- Run control with configuration management and module sequencing
- access to detector data
- access to event data.

Configuration management

All configuration files used by the offline framework are in XML format. A central bootstrap file, specified when running a program using the offline framework, contains links, as file paths or URI's, to the individual configuration files of all components.

Error checking of the configuration files is implemented using XML schema descriptions of the syntax of the configuration files. This way, most of the error checking is moved from the offline framework or client module into the XML reader. This results in simplifications in the configuration reading and checking code.

A concatenation of all the configuration files used in a run can be written out, resulting in an XML file that contains detailed record of the conditions of a given run. This file can be displayed in an XML-capable browser for examining the log file. The format of the log file is the same as that of a bootstrap file. As a consequence, one can re-create the conditions of any given run simply by feeding the log file to the auger offline main program.

Modules and sequencing

As part of parsing the configuration for a given run, the RunController part of the framework sets up the sequence of modules to be executed. The information is provided in the module sequence file. The information not only selects the modules to be pulled in for a run, but also controls the sequence in which the modules are executed. For this purpose, we define a simple XML application for sequencing. This application has two types of XML tags: a `module` tag for selecting modules to be used and a `loop` tag for grouping and loop control (see figure 2).

```
<loop numTimes="unbounded">
  <module> EventFileReader </module>
  <loop numTimes="10" save="yes">
    <!-- Generate new event from Aires/Corsika
         data stored in Event -->
    <module> EventGenerator     </module>
    <module> Simulator          </module>
    <module> EventFileExporter  </module>
  </loop>
</loop>
```

Figure 2: A simple example of a module sequence

The `loop`-tag has a few attributes that affect the way the loop is executed. Also, modules can affect looping by indicating, via return codes, that the run control should continue with the next iteration of the loop, or break out of the loop completely. The attribute `numTimes` specifies the number of iterations of a loop. The `save` attribute requests that the current event should be restored on every pass through the `loop` tag. Without saving the event, the loop continues with the event as left at the end of the previous iteration.

For more complicated sequencing, the users have the possibility to provide their own sequencing in a user-supplied main routine.

The modules have to comply to a simple interface, providing an `Init`, `Run`, and `Finish` method. A `REGISTER_MODULE` macro in the code of the module makes sure the module is known to the RunController.

Data access

The framework provides two hierarchies for accessing data: The `Detector` for access to slowly changing data like detector configuration and geometry, and calibration and monitoring data. The `Event` for the data related to a single event. Both implement parallel hierarchies that follow the detector hardware hierarchy.

The Detector

The detector (fig. 3) provides a unified interface to multiple data sources. The user sees a standard hierarchy, following the detector layout. The actual layout and organisation of the data on disk can follow the same hierarchy, e.g. in the case of an XML file, or a different one, as in the case of a relational database. A manager in the backend of the detector implementation translates between the formats. Furthermore, it is possible to have more than one manager for a given datum. This way, a special manager can override data from a general manager. For example, a user can decide to use a database for the majority of the description of the detector in a run and change a few selected pieces of information, overwriting them with data in an XML file. The selection of data sources to query is configurable via an XML file and the user code itself is unaware of the selected configuration.

The Event

The Event (fig. 4) acts as the backbone structure for communication between the modules in an offline-chain. It is set up to hold the raw and calibrated event data, augmented with Monte Carlo data from simulations and reconstruction information accumulated during analysis.

The Event structure includes a set of well-defined protocols which allow the Event to be incrementally built up by sequences of modules and interrogated at any step to discover the current constituents of the Event.

EXTERNAL PACKAGES

The Offline Framework depends on packaged written and maintained outside the offline developer team. Some of the packages are maintained by other groups in the collaboration, whereas other packages are developed and maintained externally to the collaboration. Libraries for access to raw event data, provided by the data acquisition teams, fall into the first class. External packages used in the implementation of the Offline Framework are:

- ROOT
- Boost
- Xerces-C
- CLHEP
- Aires
- Geant 4 (optional)

It is important that the number of external packages used is not too large, since a site will have to install them before they can install a fully operational version of the Offline Framework.

BUILD SYSTEM

As a build and configuration system, the Offline Framework uses the GNU tools autoconf, automake, and libtool. The advantage of using these tools is that the resulting distribution requires only readily available software which is typically installed on a *nix workstation or server (compilers, make, Bourne compatible shell, awk). Also, the resulting sequence of instructions

```
configure
make
make install
```

is well known to and understood by system administrators and advanced users.

For the user code, simple GNU makefiles are provided as part of the documentation. A shell script, `auger-offline-config`, helps to propagate information, e.g., the location of external packages for linking, from the configuration step of the Offline Framework to the user.

QUALITY CONTROL

GNU automake provides hooks for including testing, particularly as part of building a release. The Offline Framework provides a comprehensive set of unit tests that connect to these hooks. This way, we can be sure that all existing tests are run before a new release is cut and shipped to the users.

More detailed validation, especially the physics validation, will incorporated as acceptance tests into the test suite of the offline framework.

ACKNOWLEDGEMENTS

We would like to thank all our collaborators, particularly our patient testers, for their help during the development of the Offline Framework.

REFERENCES

[1] Pierre Auger Observatory homepage, http://www.auger.org/

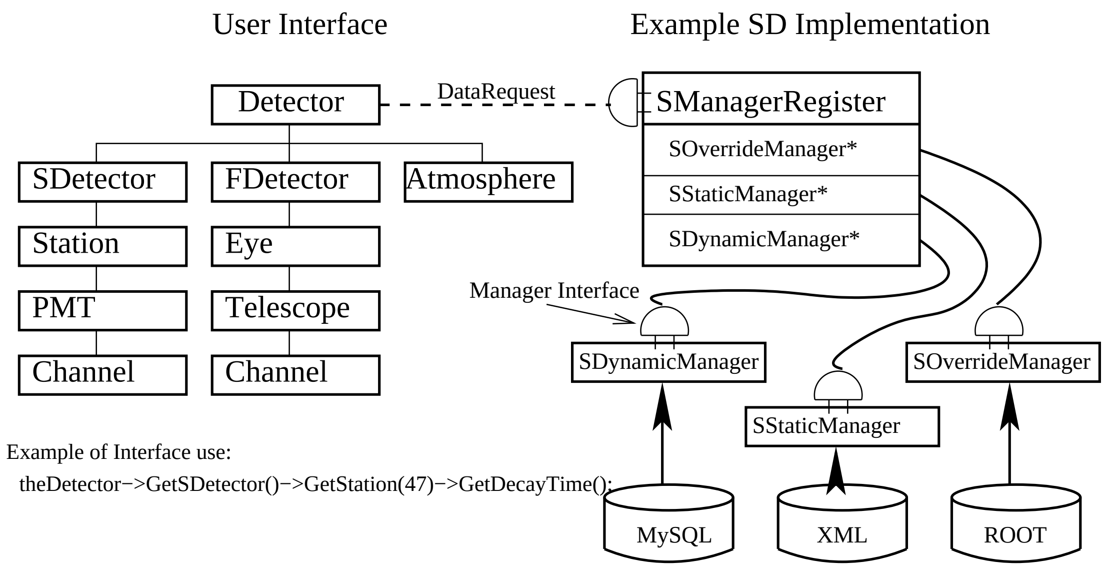
[2] The Geometry Package for the Pierre Auger Observatory, L. Nellen et. al., these proceedings

[3] The Pierre Auger Project Central Data Acquisition System, Antoine Letessier-Selvon, Auger note GAP 1999-003

[4] Auger Fluorescence Online Software, http://www-ik.fzk.de/˜mathes/

[5] Aires distribution page, http://www.fi sica.unlp.edu.ar/auger/aires/

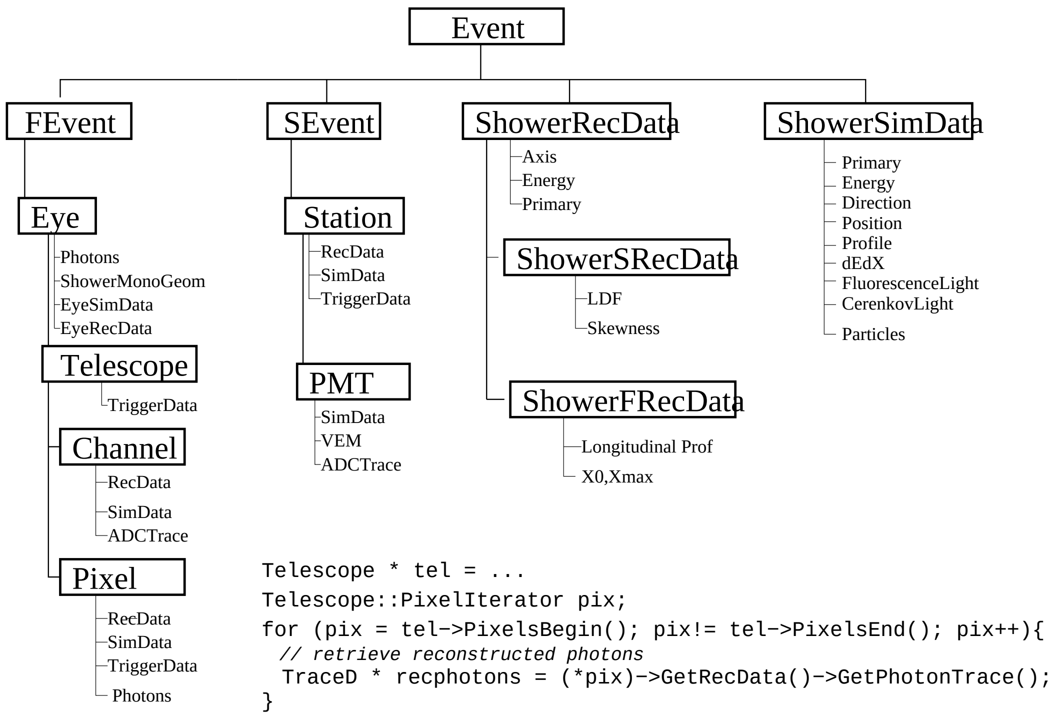
[6] D. Heck, J. Knapp, J.N. Capdevielle, G. Schatz, T. Thouw, Report FZKA 6019 (1998); http://www-ik.fzk.de/˜heck/corsika/



Example of Interface use:

```
theDetector−>GetSDetector()−>GetStation(47)−>GetDecayTime();
```

Figure 3: The detector class hierarchy with several manager back-ends



```
Telescope * tel = ...
Telescope::PixelIterator pix;
for (pix = tel->PixelsBegin(); pix!= tel->PixelsEnd(); pix++){
  // retrieve reconstructed photons
  TraceD * recphotons = (*pix)->GetRecData()->GetPhotonTrace();
}
```

Figure 4: The event hierarchy