29th International Cosmic Ray Conference Pune (2005) **00**, 101–106

FERMILAB-CONF-05-311-E-TD

The Offline Software Framework of the Pierre Auger Observatory

S. Argirò, S.L.C. Barroso, S. Dagoret-Campagne, J. Gonzalez, L. Nellen, T. Paul, T. Porter, L. Prado Jr., M. Roth, R. Ulrich and D. Veberič
for the Pierre Auger Collaboration
T. Paul (tom.paul@cern.ch), usa-paul-T-abs1-he15-poster

The Pierre Auger Observatory is designed to unveil the nature and origin of the highest energy cosmic rays through the analysis of extensive air showers. The large and geographically dispersed collaboration of physicists and the wide-ranging collection of simulation and reconstruction tasks pose some special challenges for the offline analysis software. We have designed and implemented a general purpose framework which allows Auger collaborators to contribute algorithms and configuration 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 residing in many data sources. A number of utilities are also provided, including a novel geometry package allowing manipulation of abstract geometrical objects independent of coordinate system choice. The framework is implemented in C++ and takes advantage of object oriented design and common open source tools, while keeping the user-side simple enough for C++ novices to learn in a reasonable time. The distribution system incorporates unit and acceptance testing in order to support rapid development of both the core framework and the contributed user codes.

1. Introduction

The offline software framework of the Pierre Auger Observatory provides machinery to support simulation, reconstruction and analysis work carried out by members of the collaboration. The requirements of the experiment place rather strong demands on this software. Most importantly, the framework must be flexible and robust enough to support the collaborative effort of a large number of physicists developing a variety of applications over a 20 year experimental run. It must also handle a number of data formats in order to deal with event and monitoring information as well as the output of air shower simulation codes. Additionally it is desirable to ensure that all physics code is "exposed" in the sense that any user must be able to replace existing algorithms with his own in a straightforward manner. Furthermore, while the underlying framework may exploit the full power of C++ and object-oriented design, the portions of the code directly used by physicists should not assume a particularly detailed knowledge of C++.

The offline framework was designed with these principles in mind. Implementation of the code has taken place over the last two years, and it is now being employed in analysis of data gathered by the observatory.

2. Design overview

The offline framework comprises three principal parts: a collection of processing *modules* which can be assembled and sequenced through instructions provided in an XML file, an *event* structure through which modules can relay data to one another and which accumulates all simulation and reconstruction

information, and a *detector description* which provides a gateway to data describing the configuration and performance of the observatory as well as atmospheric conditions as a function of time. These principal ingredients are depicted in Figure 1.

2.1 User code, configuration, and run control

Experience has shown that most tasks of interest to the Pierre Auger Collaboration naturally break down into sequences of well-defined processing steps. Physicists prepare such processing algorithms in so-called modules, which they can insert into the framework by adding a registration macro to their code. This modular design allows collaborators to easily exchange code, compare algorithms and build up a wide variety of applications by combining modules in various sequences.

Run-time control over module sequences is afforded through a *run controller* which invokes the various processing steps according to instructions provided in a sequencing file. We have chosen to use XML [1] to write the sequencing files as it is very easy to learn, yet grammatically rich enough to allow sequencing instruction that have enough detail to accommodate most analysis applications.

Cuts, parameters and configuration instructions used by modules or by the framework itself are also stored in XML files. A central directory points modules to their configuration file(s) by pathname or URL and creates parsers to assist in reading information from these files. The configuration mechanism can also concatenate and store all configuration files accessed during a run. This allows the creation of subsequent runs using the exact configuration employed in the original run. To check configuration files for errors we exploit schema [2] validation, in which an auxillary XML file is provided which defines a set of rules that must be obeyed by the configuration file. This saves a considerable amount of coding, since users do not have to write error checking code themselves. Furthermore, XML schema allows users to easily effect much more detailed checking than they are likely to implement on their own.

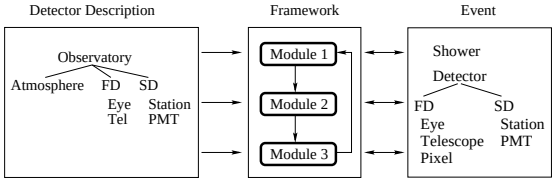

Figure 1. Simulation and reconstruction tasks are broken down into modules. Each module is able to read information from the detector description and/or the event, process the information, and write the results back into the event.

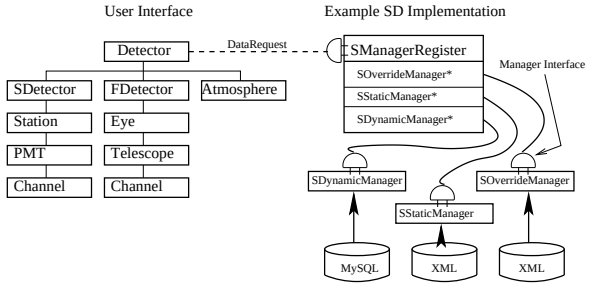

Figure 2. Machinery of the detector description. The user interface (left) relays requests for data to a registry of managers (right) which handle multiple data sources.

2.2 Event Description

The *event* data structure contains all raw, calibrated, reconstructed and Monte Carlo data and acts as the principal backbone for communication between modules. The event structure is built up dynamically as needed, and is instrumented with functions allowing modules to interrogate the event at any point to discover its current constituents.

The event representation in memory is decoupled from the representation on disk. This design choice is intended to allow one to change the underlying serialization mechanism, which maps complex objects in memory onto flat files on disk. It is prudent to maintain a design in which serialization technology can be changed without affecting existing code, as a standard serialization mechanism could be introduced into the C++ language in the future [3]. Currently, serialization is implemented using the ROOT [4] toolkit.

A set of simple-to-use input/output utilities allow users to transfer part or all of the event from memory to a file at any stage in the processing, and to reload the event to continue processing from that point onward. These utilities also support the multi-format reading and writing required to deal with different event and monitoring formats as well as the formats used by the AIRES [5], CONEX [6] and CORSIKA [7] air shower simulation packages.

2.3 Detector Description

The *detector description* provides an intuitive interface from which module authors may retrieve information about the detector configuration and performance. The interface is organized following the hierarchy normally associated with the observatory instruments. Requests for data are passed by this interface to a registry of so-called *managers*, each of which is capable of extracting a particular sort of information from a particular data source. In this way, the user sees only a single interface even though the data sought may reside in any number of different sources. Generally, we choose to store static detector information in XML files, and time-varying monitoring and calibration data in MySQL [8] databases. The structure of the detector description machinery is illustrated in Figure 2.

Note that it is possible to implement more than one manager for a particular sort of data. In 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, but override some data by writing them in an XML file which is read by the special manager. The specification of which data sources are accessed by the manager registry and in what order they are queried is detailed in a configuration file. The configuration of the manager registry is transparent to the user code.

2.4 Utilities

The offline framework is built on a collection of utilities, including a XERCES-based [9] XML parser, an error logger, and a set foundation classes to represent objects such as signal traces, tabulated functions and particles. The utilities collection also provides a geometry package in which objects such as vectors and points keep track of the coordinate systems in which they are represented. This allows for abstract manipulation of these objects, as any coordinate transformations which may be required in an operation between objects are performed automatically. The geometry package also includes support for geodetic coordinates.

2.5 Maintainability and External Packages

To help ensure code maintainability and stability in the face of a large number of contributors and a long experimental run, unit and acceptance testing are integrated into the offline framework build and distribution system. This sort of quality assurance mechanism is crucial for any software which must continue to grow and develop over a timescale of years.

Our build system is based on the GNU autotools [10], which provide hooks for integrating tests with the build and distribution system. A substantial collection of unit tests has been developed, each of which is designed to comprehensively test a single framework component. These unit tests are run at regular intervals and in particular prior to releasing a new version of the software. We have employed the CppUnit [11] testing framework as an aid in implementing these unit tests. We are currently in the process of developing more involved acceptance tests which will be used to verify that modules and framework components working in concert continue to function properly during ongoing development.

The choice of external packages upon which to build the offline framework was dictated not only by package features, support, and the requirement of being open-source, but also by our best assessment of prospects for longevity.

3. Summary

We have implemented an offline software framework for the Pierre Auger Observatory. It provides machinery to help collaborators work together on data analysis problems, compare results, and carry out production runs of large quantities of simulated or real data. The framework is configurable enough to adapt to a diverse set of applications, while the user side remains simple enough for C++ non-experts to learn in a reasonable time. The modular design allows straightforward swapping of algorithms for quick comparisons of different approaches to a problem. The interfaces to detector and event information free the users from having to deal individually with multiple data formats and data sources. This software, while still undergoing vigorous development and improvement, is now being used in analysis of data gathered by the Pierre Auger Observatory. We believe our approach and general system design may be applicable to other experiments involving large collaborations and detectors of similar complexity.

References

[1] http://www.w3.org/XML
[2] http://www.w3.org/XML/Schema
[3] see for example http://www.boost.org
[4] http://root.cern.ch
[5] S. Sciutto, AIRES User's Manual and Reference Guide, http://www.fisica.unlp.edu.ar/auger/aires
[6] T. Pierog *et al.*, to appear in proceedings of 13th International Symposium on Very High-Energy Cosmic Ray Interactions at the NESTOR Institute, Pylos, Greece, 6-12 Sep 2004, [arXiv:astro-ph/0411260].
[7] D. Heck, J. Knapp, J.N. Capdevielle, G. Schatz, T. Thouw, Report FZKA 6019 (1998).
[8] http://dev.mysql.com
[9] http://xml.apache.org/xerces-c
[10] http://www.gnu.org/software/autoconf
http://www.gnu.org/software/automake
http://www.gnu.org/software/libtool
[11] http://cppunit.sourceforge.net/doc/1.8.0