Journal of Physics: Conference Series **396** (2012) 022045 doi:10.1088/1742-6596/396/2/022045

The Offline Software Framework of the NA61/SHINE Experiment

Roland Sipos[1], Andras Laszlo[2], Antoni Marcinek[3], Tom Paul[4], Marek Szuba[5], Michael Unger[5], Darko Veberic[6] and Oskar Wyszynski[3]

[1] Eotvos Lorand University Budapest, HU
[2] Wigner Research Center for Physics, HU
[3] Jagiellonian University, PL
[4] Department of Physics-Northeastern University, US
[5] KIT - Karlsruhe Institute of Technology, DE
[6] University of Nova Gorica, SI

E-mail: Roland.Sipos@cern.ch

Abstract. NA61/SHINE (SHINE = SPS Heavy Ion and Neutrino Experiment) is an experiment at the CERN SPS using the upgraded NA49 hadron spectrometer. Among its physics goals are precise hadron production measurements for improving calculations of the neutrino beam flux in the T2K neutrino oscillation experiment as well as for more reliable simulations of cosmic-ray air showers. Moreover, p+p, p+Pb and nucleus+nucleus collisions will be studied extensively to allow for a study of properties of the onset of deconfinement and search for the critical point of strongly interacting matter.

Currently NA61/SHINE uses the old NA49 software framework for reconstruction, simulation and data analysis. The core of this legacy framework was developed in the early 1990s. It is written in different programming and scripting languages (C, pgi-Fortran, shell) and provides several concurrent data formats for the event data model, which includes also obsolete parts.

In this contribution we will introduce the new software framework, called Shine, that is written in C++ and designed to comprise three principal parts: a collection of processing modules which can be assembled and sequenced by the user via XML files, an event data model which contains all simulation and reconstruction information based on STL and ROOT streaming, and a detector description which provides data on the configuration and state of the experiment. To assure a quick migration to the Shine framework, wrappers were introduced that allow to run legacy code parts as modules in the new framework and we will present first results on the cross validation of the two frameworks.

1. Introduction

SHINE stands for SPS Heavy Ion and Neutrino Experiment [1]. Its physics goals are precise hadron production measurements for improving calculations of the neutrino beam flux in the *T2K* neutrino oscillation experiment as well as for more reliable simulations of cosmic-ray air showers. Furthermore it searches for the critical point and the onset of deconfinement of strongly interacting matter in ion-ion collisions. A two dimensional scan of the phase diagram of strongly interacting matter will be done by changing the ion beam energy at the SPS (13A - 158GeV) and the size of the colliding nuclei. The critical point would be indicated by a maximum in the fluctuation of the particle multiplicity and other physical observables. The onset of

deconfinement is revealed by rapid changes in the hadron production properties. The NA61 experiment is located at the *CERN SPS*, and consists of several types of detectors. The main tracking detectors (Time Projection Chambers) were inherited from it's predecessor, the *NA49* experiment[2]. In addition, several new detectors were developed and installed. The setup has unique and outstanding capabilities in order to achieve the following requirements:

- large acceptance,
- high momentum resolution,
- high tracking efficiency,
- high event rate.

Along with a large part of the hardware, also the track reconstruction software was inherited from NA49. It is based on a dedicated tool, called `DSPACK`[3]. It is an implementation of a client-server software architecture with several functionalities to support offline related applications. It contains combinations of early object oriented principles in order to be able to create unique data sets and store the structures in the memory. With the help of an interface, `DSPACK` makes it possible for data handlers to dynamically define objects from data files. The use-cases became really wide as definitions are able to contain needed run-time options and parameters. Also connection of standalone clients to a `DSPACK` server instance is supported, and thus these can manipulate the data. A chain of clients represents offline applications such as simulation, calibration, reconstruction or data analysis chains. Every standalone client handles a subtask of the main task flow.

2. Software upgrade proposal

Patches and modifications in order to support NA61 requirements within the legacy NA49 software lead to several unpleasant side effects. Apart from the general difficulty of working on obsolete systems, the existing software is almost impossible to maintain properly in its present form. The problems that needed specific attention are the following.

- **Portability issues**: The legacy software is bound to the CERN computational infrastructure and uses several dependencies and external data tables. It is simply not possible to install the software on a personal computer and with this, do not have the option to work remotely without network connection.
- **Concurrent data formats**: The collaboration uses DSPACK format to store reconstructed events but prefers the more NA61 specific *DST* format based on ROOT for physics analysis. Continuous cross conversions in different cases makes the cooperative work harder.
- **Mixed programming languages**: The software was implemented using several programming languages such as Fortran, C and C++. The steering routines are sequences of client calls and DSPACK instructions and therefore also extensive use of scripting languages is present.
- **Obsolete software architecture**: The unique memory management tool DSPACK is not widely used in the HEP community. Beside the lack of support, the implementation contains several hidden bugs and misleading parts.
- **Outdated production tools**: The used simulation framework is implemented in GEANT3, which does not support new supplementary detectors in the experiment, therefore porting to GEANT4 is necessary. Improvement of the used reconstruction techniques is also required.
- **Lack of support**: The used software is implemented via outdated design patterns and languages. Large scale involvement of people into such developments is inherently difficult. The existing documentation has missing parts and even contains incorrect elements.

The proposed upgrade considerations [4] are the following.

- **Unified language and data format**: We propose to use a unified programming language, with the choice of C++. The language is widely used in the HEP community and we think that it is easy to acquire at least a user level knowledge.
- **New design**: The new software should use the framework philosophy, in particular the skeleton of Offline Software Framework of the Pierre Auger Observatory[5].
- **Integration of legacy software**: The framework needs to contain the legacy software to provide a better supervised environment for it and to start production as soon as possible.
- **New offline algorithms**: Outdated algorithms that do not satisfy NA61 needs should be reimplemented in the new software framework.

3. The Shine $\overline{\text{Off}}\underline{\text{line}}$ Framework

Based on the known problems the collaboration decided to implement a software upgrade to replace the obsolete legacy package. The skeleton of the $\overline{\text{Off}}\underline{\text{line}}$ Software Framework of the Pierre Auger Observatory was used for the Shine $\overline{\text{Off}}\underline{\text{line}}$ Software Framework. The design contains important key features, such as extensibility, and there are three principal pillars on which the planned framework is built.

- **Modules**: Basically modules stand for processing elements, and provide the user interface to the framework. They should be assembled and sequenced via configuration files. These will represent the application programming interface (API).
- **Event**: The global data model which stores offline related information, is the event. Modules can acquire data from the event container, and only if needed, they communicate through this structure.
- **Detector description**: The experimental facility needs to be modelled and implemented in the detector description framework. This should be the sole provider of the updated setup and calibration data that correspond to the detector configuration.

3.1. Configuration

Central configuration provides the interface to every configuration related task. The goal is to provide access for users to configuration files via the *CentralConfig*, and give option to read, write and modify settings of the detector description, data or modules. The files are written in XML and the framework supports run-time XML schema validation and MD5 check-sums for these. The *CentralConfig* is configured via so-called *bootstrap* files that contains *example*, *standard* and unique configurations. Parameter overrides and schema path replacements are also supported for consistency.

3.2. Detector description

One of the principal parts in experimental physics software is the way in which the detector is modeled and the interface works in order to access instrument related data. This needs to be separated into two important sections, the *Detector User Interface* and the *Detector Backend*. The schematic of the detector description is shown in fig. 1.

Users deal with a single interface that is extensible and follows the hierarchical structure of the physical detector. Users are not allowed to modify the provided data, so they get constant references to the fetched information. The backend handles the updating and acquiring of data from different sources. Requested data can come from a wide range of formats, each handled by it's unique manager interface. Usually static information is stored in XML (Xerces wrapper), dynamically changing data should be stored in database systems (MySQL wrapper).

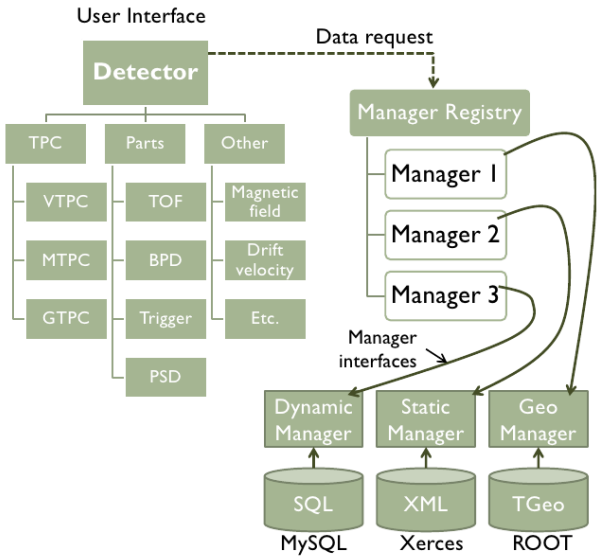

Figure 1. Schematic of the detector description in Shine.

3.3. SHOE - The Shine Offline Event

The aim of SHOE is to replace the legacy DSPACK, ROOT61 and NDST structures by one unified data format. It is based on ROOT and made of a streamable collection of classes for offline purposes. The content of information is scalable and therefore supports different levels of detail. One can access parent- and child-objects using simple indices (SHOE-Laces), and the connection between event objects is made by standard lists. With this method the navigation through the data is easy and allows fast random access and removal of objects. An example Ξ particle described with the SHOE event structure is shown in fig. 2.

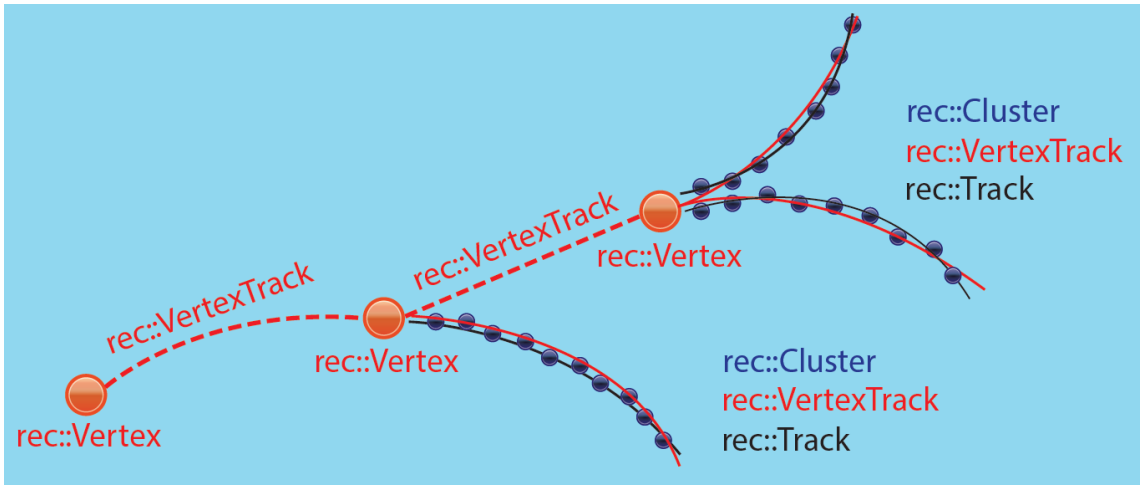

Figure 2. Example usage of the new event structure SHOE.

3.4. Modules
The elements into which users can insert processing steps inside the framework and manipulation of the event structure is the Module. The interface called *VModule* defines three virtual functions that users need to implement. Each method returns a result flag in order to report success, failure or even looping instructions for the *RunController* that traces and monitors the work-flow of the offline application. Primary tasks (application) are sequences of sub tasks (modules), supervised by the mentioned *RunController* element. The user defined functions are the following.

- **Init**: This function is invoked at the beginning of each run, supervised by the *RunController*. Generally this function implements needed preliminary settings before the run for the module, and is to be done only once for initialization.
- **Process**: The method contains the processing code itself that should be done once for each event. A reference to the *Event* structure is passed, so the data manipulation happens in this function.
- **Finish**: When a run is finished the module's *Finish* process is called, usually for closing files, writing data such as histograms or remove objects.

The returned *ResultFlag* shows the current outcome of each function. Basically this serves as the communication between the framework and the processing modules.

4. Support
The framework comes with many additional features and support to aid users. Some of these tools are essential for the proper working of the framework, others are meant to help implementations by users or prevent code duplication.

4.1. Utilities
A wide range of utilities from the field of mathematics, physics and computer science helps the work of the users. These are frequently used in modules (e.g.: ODE Runge-Kutta Integrator) and also in the core framework (e.g.: XML Reader, custom exceptions and shadow pointer).

4.2. Dependency control
A tool called Shape was invented to download and install every external dependency with ease and to set up the required running environment for the Shine Framework. As one of the main requirements for the Shine Framework was it's portability, it was essential to provide such a tool with the framework.

4.3. Automatization
A buildbot was set up for automatizing compilation, testing and validation after each commit and inform developers about integrity. Users should also provide test procedures for their implemented modules to ensure the proper performance. Doxygen documentation was generated from the source code in order to provide a clear picture of the framework and give information for Shine developers and power-users.

5. Summary
The new Shine $\overline{\text{Off}}\underline{\text{line}}$ Software Framework for the NA61/SHINE Experiment provides a user friendly, stable and flexible software system. It uses state-of-art techniques of computer science and satisfies most of the NA61 offline analysis needs. Huge efforts were made to revive and integrate the legacy software within the new framework to cross validate the two system. The continuous migration of collaboration members to the Shine Framework shows, that a reliable and extensive framework is rather a need, than an optional choice for experimental physics experiments.

References

[1] N. Antoniou et al. the NA61/SHINE Collaboration 2007 *CERN-SPSC-2007-004, CERN-SPSC-2007-019*; N. Abgrall et al. the NA61/SHINE Collaboration 2008 *CERN-SPSC-2008-018*.
[2] S. Afanasiev et al. the NA49 Collaboration *The NA49 Large Acceptance Hadron Detector*; *Nucl. Instrum. Meth. A430 (1999) 210*.
[3] R. Zybert, P. Buncic, *DSPACK: Object manager for high energy phisics*; *Proc. CHEP95 (1955), 345*.
[4] Review of NA61 Software Upgrade Proposal,
`https://indico.cern.ch/conferenceDisplay.py?confId=125760`
[5] S. Argiro et al. *The Offline Software Framework of the Pierre Auger Observatory*; *Nucl. Instrum. Meth. A506 (2003) 250*.