D 11.5: The platform for Monte-Carlo simulations

Task 2 of the SiNuRSE Joint Research Action deals with the development of a virtual Monte-Carlo platform, a framework where the nuclear physics community can perform simulation and data analysis. The activities involved in this task are geared to serve the development, exemplification, documentation and dissemination of a simulation and analysis tool for the whole community.

The adopted solution, the EnsarRoot framework for simulation and data analysis for ENSAR-SiNuRSE, uses the VMC facilities in ROOT to interface with particle tracking engines, together with the complete ROOT libraries for data analysis and visualization.
In this report, we summarize the main characteristics of the framework, including the arguments leading to the selection of the framework, the improvements done for the parameters database management, and some aspects to the detector integration and quality assessment.

The EnsarRoot code is available for downloading (http://igfae.usc.es/~sinurse/) including installation guides and documentation.

EnsarRoot: a framework for simulation and analysis for SiNuRSE

EnsarRoot, the framework for simulation and data analysis for ENSAR-SiNuRSE, is based on the FairRoot framework (http://fairroot.gsi.de/) which is fully based on the ROOT[1] code. The user can create simulated data and perform analysis within the same framework. Moreover, Geant3 and Geant4[2] transport engines are supported; however, the user code that creates simulated data does not depend on a particular Monte Carlo engine.

The EnsarRoot framework delivers base classes which enable the users to construct their detectors and analysis tasks in a simple way. The code derives from the general classes of FairRoot, adding and specifying the geometrical, physical description and response of the detectors of selected example setups, database support, event generators for the reactions of interest and analysis and event visualization tools. It expands the framework behavior by including a dedicated physics list for low-energy neutrons, gamma interactions and nuclear fragment transport while supporting database connectivity to handle multiple experimental setups.

[1] Available at: http://root.cern.ch
[2] Available at: http://geant4.cern.ch/

The selection of the FairRoot framework to derive EnsarRoot has been the result of a careful evaluation, taking into account that the FairRoot base library provides:

- A common data structure for simulation and analysis based on efficient and well-tested data arrangements (TClonesArray, TFile, ...) from the ROOT system.
- A common geometry description based on the ROOT Geometry Modeller.
- An interface to different Monte Carlo engines (Geant3, Geant4, ...) using the ROOT Virtual Monte Carlo package.
- Detectors base class handling initialization, geometry construction, hit processing, event loop management, tasks management, ...
- Geometry input readers supporting ASCII, ROOT and STEP (CAD) formats.
- A generic event display based on Eve and Geane.
- A Runtime database for geometry and parameters handling.
- A Fast simulation base services based on Virtual Monte Carlo and the ROOT Tasks library.
- ROOT macro commands for steering the simulation and the analysis.
- ROOT macro commands for configuring the different Monte Carlo engines.
- Developers tools: git support, CMake employed as a flexible build system for different platforms, CDash and CTest for Code Quality Assessment...
- Support for grid computing using the Alien Grid Middleware.

The EnsarRoot code derives from the general classes of FairRoot, expanding their behavior with elements which are relevant for the experiments performed in our community and examples and templates of different detectors and setups which could serve as a base for the development of their own detector arrangements. It extends the framework behavior by adding a dedicated physics list for low-energy neutrons, gamma interactions and nuclear fragment transport while supporting the new parameters database to handle multiple experimental setups and different databases through SQL.

The present implementation of EnsarRoot contains examples for scintillating detectors (*scintillator* and *scitof* modules), full calorimetry (CALIFA-R3B in *calo* module), silicon detectors (*Silicon* module), an RPC-based cosmic rays detector (*tragaldabas* module), and a setup for ring experiments (*E105* module). More information as well as how-to documents to implement new setups is also available (http://igfae.usc.es/~sinurse/).

Parameter database using "FairBase" framework

Within the FairRoot framework, data IO as well as parameter handling and database connections are handled by the framework. In this project, the database interface and parameter handling interfaces of the framework are extended and modified in such a way that the detector parameters could be written to and read from an SQL database. For this purpose a number of new interfaces, library functions and algorithms are

implemented and integrated inside the framework. Finally, the simulator code written for ENSAR E105 experiment is adapted as an example to demonstrate the functionality of this framework and to serve as a starting point for the future development of new simulation, and data-processing programs.

We have extended the FairRoot framework to provide database communication via the ROOT TSQL interface. This ROOT facility provides the opportunity to use different DataBase Management Systems (DBMS) without changing the user code and writing DBMS dependent programs. At this moment MySQL[3], Postgresql[4] and Oracle[5] DBMS are supported by TSQL interface in different states. For the current implementation, we have chosen to use MySQL as the testing and development platform. The reason is that MySQL is available for different operating systems and is quite easy to install and deploy. The older versions of FairRoot framework could write into and read from parameter files, both in ASCII and root format. In order to support SQL based IO, a new interface has been designed and implemented.

The design of the interface is done in a way that the details of the communication and the read/write operations are fully hidden from the end user. In the case of a detector code developer, it means in practice that the parameter handling code for each detector or sub-detector needs to be extended by a small number of procedures in order to use the current SQL IO facilities. These are procedures/functions to describe SQL structure of the parameter container, setting (Filling), storing and fetching the container data from the database.

At this moment, the SQL IO interface supports basic parameter types such as integer, float, double and character. It is planned to extend the functionality to store even complex data types in the form of Binary Large OBjects (BLOB). The latter could be useful for storing an array of data or even histograms. In case of an end user, one can continue using the same macros and programs for storing the data into or fetching the data from a database; one simply needs to specify the address of the database instead of a parameter file name. In the latter case, one needs to have read and/or write access to the chosen database. Furthermore, the current implementation supports multiple databases from which the parameters can be fetched or they can be written to. These may be specified by a number of shell environmental variables.

For further information and details of the implementation, we refer to the FairRoot documentation and source files available at GSI web and source repository[6].

---

[3] Available at: http://www.mysql.com/
[4] Available at: http://www.postgresql.org/
[5] Available at: http://www.oracle.com
[6] GSI Helmholtzzentrum für Schwerionenforschung GmbH. http://www.gsi.de
https://github.com/FairRootGroup/FairSoft

In order to provide a working example we have chosen to adapt the EnsarE105 (E105 was a ring experiment recently performed at GSI) simulator in such a way that it is capable of using the FairRoot SQL IO facility. This implementation, in combination with the other examples provided in the "fairbase" implementation, could be used as a starting point for new data production and analysis chain implementation. The E105 source directory contains the detector geometries, event generator and example macros. For more information, see the EnsarRoot documentation.

As future work, we point out a number of features which are not included in the framework yet but could be possible extensions of the current implementation[7].

- In the current state, the FairRoot database interface is designed to handle primary data types like integer, float, double and characters. It is planned and desirable to extend the interface to handle more complex types such as arrays of a primary type.
- Database replication facilities. To replicate data consistently and maintain the data integrity across different servers and client applications.
- A full object decomposer tool chain in the data-handling interface to store and retrieve more complex objects by means of their members.
- A complete SQL translator between different SQL versions will help to create more efficient and optimized queries.
- A uniform administrator tool set via FairBase tool.

Detector integration and quality assessment

As discussed in the introduction, the general framework EnsarRoot contains modules which can be used as examples or test cases, or taken as templates for the development of different setups.

-Geometry, hits and digitization developments:

As a starting point, a novice module, we have chosen two template detectors: a scintillator and a silicon detectors (template_scintillator and template_silicon modules in EnsarRoot). The geometry is developed following the TGeo geometrical modeller (a combination between a GEANT--like scheme and a normal CSG binary tree at the level of shapes) of ROOT package. The mother/daughter concept is preserved on all detector geometry descriptions. Practically, every geometry defined in GEANT style can be included or implemented in the framework. TGeoManager is responsible for building and tracking geometries. It contains lists of media, materials, transformations, shapes and volumes. Logical nodes (positioned volumes) are created and destroyed by the

[7] For the most recent status and plans, please contact the FairRoot developers.

TGeoVolume class. They are constructed following general rules: volumes need media and shapes in order to be created, both container and volumes must be created before linking them together, and the relative transformation matrix must be provided. The documentation shows how to construct a detector from each template type.
The Monte-Carlo tracker interactions can be accessed through a class collecting the hits produced (using the class TClonesArray from ROOT). The contents of the array, the collection of interactions, are filled by proper methods in the template detectors class.

Next, detector digits, essentially a hit in the sensitive detector storing the main detection characteristics (energy, time, …), are produced by a set of tasks which run on an event-by-event basis. The resulting collection of the digits, stored as before in TClonesArray structures, can be analyzed by additional tasks up to any level of elaboration. The template detectors also include container parameter classes (EnsarScintillatorContFact, EnsarScintillatorGeoPar, EnsarScintillatorDigiPar, ...) which could control the parameters in the geometrical construction, digitization, calibrations, ...

-Event display and detector display examples for different detectors:
The purpose of this task was, on the one hand, to identify the general elements and construct classes for events and hits visualization and functions for specific requirements (possibility for cuts and selections) and, on the other hand, to develop the components for construction of display examples. The event display would serve not only in the visualization of the reconstruction process (showing hits and times on the raw level and relating it to the particle hits during the reconstruction, on an event-by-event basis) but also during the online event display.

The Visualization Classes that we have proposed to develop allow to manage the visualization of geometries, trajectories and hits. These classes can have multiple uses: as elementary blocks for visualization tasks with simple complexity, as reference -classes for visualization classes with more advanced complexity, or simply as examples of visualization usage. Demonstration classes for the event viewers in the CALIFA and the template scintillator and silicon detectors are included in EnsarRoot.

The object-oriented classes for visualization were developed taking into account that existing graphical displays already contain much of the basic functionality that we need without much additional coding: rotate and zoom; click and inspect objects on canvas, display using OpenGL[8], etc. However, documentation of these classes is requested since all the existing classes are very poorly documented. Construction of display classes was done following the next steps:

1. Identification of common elements of event display with other systems (e.g. monitoring and analysis systems) in order to avoid duplication of effort. The

[8] https://www.opengl.org/

ROOT class TEveGeoNode provides a simple and natural means of building detector geometry and placing hits within that geometry.

2. Construction of generic graphics classes for detector and event objects.
3. Development of components for graphical user interfaces for archetype detectors.
4. Documenting classes for different archetypal detectors (CALIFA, scintillators and silicon in particular) which demonstrate the usage of Event Display classes.

-**Quality Monitoring:**

Quality Monitoring (QM) is an important aspect of many Nuclear Physics experiments, where the detectors employed are increasingly sophisticated devices. Developing an integrated system for monitoring detectors key parameters has become an important task. The QM framework is built for the purpose of checking the quality of the data at different levels of data acquisition (data from Monte Carlo simulations and real experimental data).

The QM is performed in three steps:

- At the first step, QM data objects for a given level and a given detector are stored in a TList type class (EnsarQM). Here, the global QM class tree structure – EnsarQM.cxx – will have a detector list and output file (QM.root).
- For each detector class, build quality-control histograms classes are created (a template for a scintillator detector and a silicon detector in this case). At this step, QM data objects derived from the ROOT TH1 histogram class are built (classes EnsarScintilatorQMData.cxx and EnsarSiliconQMData.cxx).
- The last step is to build a checking class for the different QM parameters with user-defined data.

The structure of QM classes has been completed. Integration of QM in EnsarRoot repository is on the way.