RootSorter: A New Analysis Framework for ANKE

V. Hejny, M. Hartmann, A. Mussgiller

The current analysis software (sorter), which has been used during the last years, uses the popular CERNLIB package and, thus, PAW as the central user interface. The development of these libraries at CERN had been stopped some time ago and the design of new, modern packages has been started. One of these projects, namely ROOT [1], has been widely accepted as a direct successor to PAW. ROOT is written completely in C++ and provides a large number of useful classes for physics analysis. Moreover, the macro language is C++, too, which allows to test code in interpreted mode first and compile it for fast production runs. Thus, it was quite natural to use the ROOT package as a basis for a new analysis project.

Basic ideas

Some design goals were fixed to make the new software more convenient for the regular user than the old sorter was. These are:

- **Modularity.** It should be a simple task to add new analysis procedures to the code without changing anything in the package.
- **Automatism.** The user should neither be responsible for calling initialization functions of library packages nor should he/she take care about all necessary calls for data processing in each event (some people would call this *plug'n'play*).
- **Central database.** All persistent analysis parameters (i.e. run parameters, calibration data, geometry information, etc.) should be stored in *one* central database. It must be possible to review parameters for any point in the past.
- **Support of ROOT I/O.** ROOT trees[1] provide an easy-to-use interface for data handling. It must be possible to write out any raw data or intermediate and final results to such a tree.
- **Remote access.** We decided to build a client/server mechanism (using TCP/IP) into the analysis: the process, which does the actual data processing is completely separated from the display process. This has the advantage that the display process (i) can run on any host and (ii) is able to display results from various analysis processes in parallel.

Analyzing data

There are basically two central entities the regular user has to deal with: objects, which contain the experimental data, and modules, which contain the analysis code. Consequently, they are called *data objects* and *analysis modules*. For both the software package provides appropriate base classes, which already do most of the administration work. The user can concentrate on the important part, i.e. physics. As nearly everything within the RootSorter framework, these objects have unique names. For example, if a user wants to know whether there is a K^+ in the ANKE telescopes, all he/she needs is the name and the class name of the (standard) analysis module, which does the kaon identification[2]. This is enough information to access the module. The user does not need to take care about the creation of this object (this will be done automatically, when it is requested the first time), nor does one need to call any initialization functions. The latter is taken care of by the author of the analysis module in the constructor of the class. This reflects one of the basic strategies: an object itself knows what it needs and what it has to do. Thus - in this example - the kaon analysis knows best what it needs for kaon identification and, therefore, asks for subsequent analysis modules, e.g. time-of-flight analysis, momentum reconstruction, etc. The modules at lowest level will then ask (also by name) for the data objects, which hold the necessary raw data. These data objects are filled from several data sources. At the moment these are EMS data streams and ROOT trees.

[1]Extended ntuples, see [1].

[2]Since the development of standard analysis modules is still going on, this specific high-level analysis module does not exist yet.

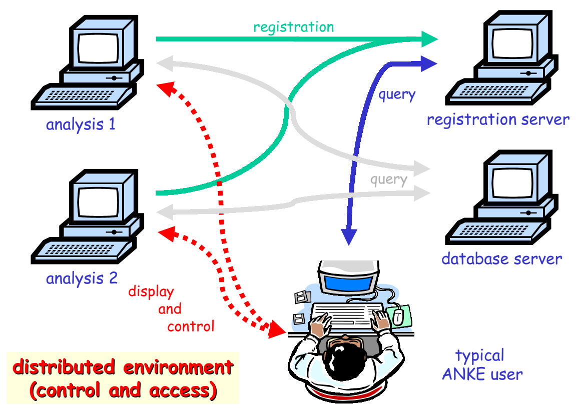

Fig. 1: Distributed environment of RootSorter. All analysis programs can be accessed from any place.

For each event, this whole procedure is triggered only on user request. As long as nobody asks for kaons, the kaon analysis is not executed at all. Consequently, the underlying low-level modules are not called and the data objects are not filled. Everything is done *only* on demand.

Some standard analysis modules (at the moment mainly those, which are filling raw spectra) are distributed with the code. To use one of them, a one line statement has to be added to the user's main program. A pure raw analysis is part of the package.

Parameter database

A crucial point of any reproducible analysis is the consistent usage of general analysis parameters. It has to be assured that anyone uses the correct (and for each run unique) values for target and detector position, beam settings, counter calibration, etc. Therefore, all these parameters should be stored at a central place and everyone is forced either to use this canonical set or to argue for specific changes.

RootSorter works with a central MySQL database (using the interface provided by ROOT). Different types of parameters are stored in different tables and each entry (i.e. line) in a table is treated as one *data set*. For each table a special C++ class exists, which can hold the values of one row. Herein, each column corresponds to one member variable in the class. To make life as easy as possible, automatic streamers to and from the database are provided. Everyone, who wants to introduce a new type of parameters, just has to define the class with the appropriate variables. Nearly everything else is taken care of automatically.

In addition to the user-defined data, each data set has five more entries: a class version number, a unique hash value to identify a data set and three dates. Two of them - giving a start and stop time - describe to which experimental period the data belong. Normally these dates are taken from a database table containing all run information and the user has to name the run number only. The third date tells at which time the data set became invalid or will become invalid (for a valid set, this date is set to a far future). The general idea is *not to delete any data set*. With this scheme it is possible to ask for a set, which had been used, for example, one year ago. This scheme was triggered by a description of the HADES database [2] at GSI.

Using this information, the SQL interface can be asked to return the entries of a specific table, which belong to a certain run period and are valid at a certain date (normally the actual date). To cover this by an even more user-friendly mechanism, the creator of a parameter table has to provide a interface class (called *parameter object*), which can be asked to return a certain value stored in this table, e.g. the position of a certain counter in the forward hodoscope. Again, just asking for this object will trigger everything else. All complicated things are hidden from the regular user.

For those, who have no online connection to the central database server, there is the option to download the database (either full or in part) as ascii tables and use them instead of the SQL database by setting one run-time option. A third mode, mainly used for parameter import and export, are XML files. Here, either a predefined or a user-defined layout can be used.

Histogramming

In every analysis module, which fills histograms in the event loop, these histograms can be created according to the normal ROOT syntax. In addition, the author of the code has to register them within the framework. With the corresponding function calls also a *directory* is passed, where the user can find a histogram later. On request, this directory structure will be sent to a client session (see below) and is used to retrieve single histograms on demand.

Remote control

To allow permanent access from "outside", the analysis runs in threaded mode: the main thread, the event loop and a communication thread. The latter listens on a free TCP/IP port for incoming client connections. For each connection a new thread is started to handle the client requests.

To make the outer world know on which port an analysis awaits an connection, one of the first tasks of the analysis program is to communicate this information together with its name and the computer name, where it's running on, to a central daemon process. This process is normally running on a fixed computer.

The client session

To be able to "talk" to an analysis process, one has to start a ROOT session and load the RootSorter client library. One basic function in this library connects to the central registration daemon and retrieve a list of running analysis programs. The user can select one and fetch the full list of histograms available for this analysis. This list is put into a new top-level folder, which can be accessed, for example, with a TBrowser

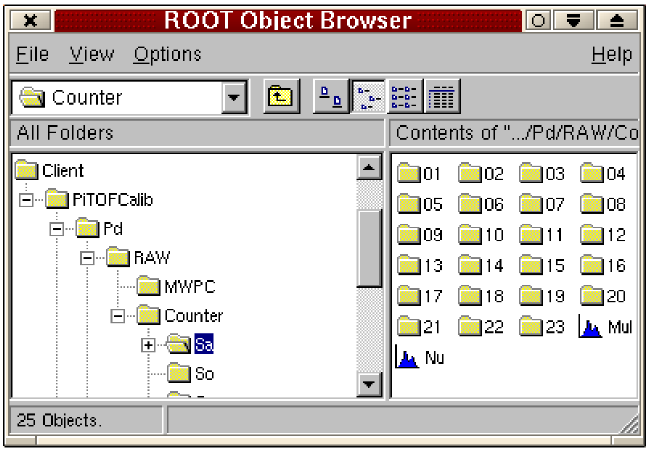

Fig. 2: Histogram folder of the analysis PiTOFCalib. The folder containing the raw spectra of the start counters in the telescopes is opened. There are two histograms in (multiplicity and pattern) and 23 more subfolders.

object (fig. 2). The easiest way to display a histogram is to double-click on the icon - the same way you would use for local histograms. Note: You will get automatically the most actual version of the histogram!

There is also a more convenient way to display one or a set of histograms. One can define control panels assigning a page layout with histograms to a button. Thus, with one click one can display the energy loss in a whole bunch of counters, for example.

Displaying histograms is only one feature of the analysis-client connection. It is also possible to start, stop and abort the analysis, to load and save histograms to file, to clear them and to send commands to different analysis modules, etc. All operations, which influence the analysis itself, can be protected by a password to avoid accidental changes by other users.

Status and outlook

All of the features described here are already implemented. Most of the recent data sets (the tests of the new spectator setup, some drift-chamber tests [3] and the Φ [4] and η [5] beam times) are analyzed within this framework. Therefore, one can expect that the number of available standard analysis modules will grow soon. In addition, the on-going work has to focus on the central parameter database, which is still quite empty. One main goal is to store the ANKE geometry in this database and to build a dedicated interface based on the new ROOT geometry package. Moreover, storing the parameters of all analysis modules (like time-of-flight cuts, etc.) is as much important. However, the latter can only be successful, if everyone, who contributes with an analysis module, will also provide the correct parameter interface.

References:

[1] http://root.cern.ch, CERN.

[2] http://www-hades.gsi.de, GSI.

[3] A. Wrońska et al., Test of Drift Chambers at ANKE, contribution to this annual report.

[4] M. Hartmann et al., First Observation of K^- and Φ mesons at ANKE, contribution to this annual report.

[5] N. Lang et al., Measurements on the Reaction $p+d \rightarrow d+p_{spec}+\eta$ at ANKE, contribution to this annual report.