MARaBOU - A MBS and ROOT Based Online/Offline Utility

R. Lutter, O. Schaile, K. Schöffel, K. Steinberger, P. Thirolf,
and C. Broude[1]
Accelerator Laboratory of the Universities of Munich, D-85748 Garching, Germany
[1]Weizmann Institute of Science, Rehovot, Israel
email: marabou@physik.uni-muenchen.de

Abstract

MARaBOU is a system for data acquisition and evaluation developed at the Tandem Accelerator Laboratory of the Universities of Munich. It consists of a front-end system for data readout, event building, and data transport based on the Multi Branch System MBS, and a back-end system responsible for setup, run control, histogramming, data analysis, and data storage written within the ROOT framework. The ROOT part includes a GUI to control the MBS front-end, a histogram presenter to visualize the data, and class libraries to describe the experiment.

I. Introduction

MARaBOU [1] has been recently established as the common data acquisition system at the Tandem Accelerator Laboratory of the Universities of Munich. MARaBOU consists of two main parts as shown in fig. 1: it uses the Multi Branch System MBS [2, 3] developed at the Gesellschaft für Schwerionenforschung (GSI), Darmstadt, Germany, as a front-end for data readout, event building, and data transport; the back-end system is written within the ROOT framework [4] and provides class libraries for setup, run control, histogramming, and data analysis and storage. A special code generator eases the use of the system and helps the user to start from scratch.

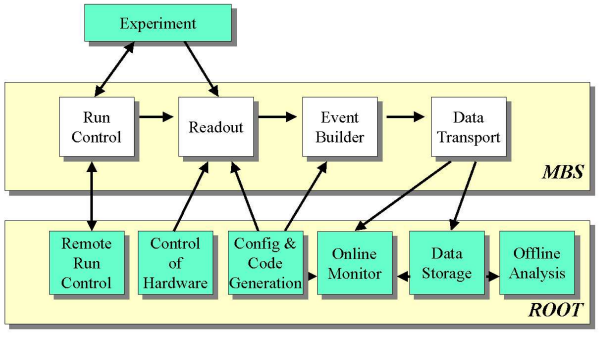
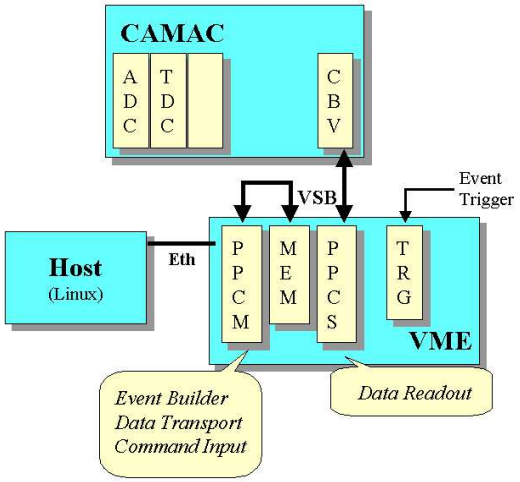


Figure 1: MARaBOU tasks

II. Hardware Configuration

Fig. 2 shows the current hardware configuration. Each data acquisition station is equipped with a pair of PowerPCs (CES RIO2-8062, 200 MHz) and a VME-CAMAC interface (CBV, Bastian Technology). A trigger module (GSI) synchronizes the data readout. Data are read from CAMAC by one processor via a VSB connection and written to a dual-ported memory, the second processor then takes data from the VSB port of the memory and sends blocks of formatted events to the Linux host.



PPCM PowerPC **Master**, 200 MHz, LynxOs
PPCS PowerPC **Slave**, 200 MHz, LynxOs
MEM Dual-ported Memory
VSB VME Subsystem Bus
CBV VME-CAMAC Interface
TRG Trigger Module

Figure 2: Hardware configuration

III. Code Generation

The process of code generation is shown in fig. 4. The user has to describe the event structure of his experiment by writing a few C++ statements to a so-called ROOT macro (fig. 3). These declarations are then processed by ROOT's C++ interpreter CINT [5] and generate code for the MBS front-end as well as for the analyzing program in the Linux workstation.

The MBS part has to be compiled under LynxOs and to be linked to the MBS readout task. It performs CAMAC and VME readout and formats events according to definitions in the configuration macro.

The analyzing part consists of a set of C++ class definitions and methods providing means to process incoming event data. It includes standardized methods for histogramming, monitoring, and event storage in ROOT tree format. The user then has to overwrite the Analyze() method with his specific event analysis. Compiling this code and linking it to ROOT and MARaBOU libraries results in a module called M_analyze which acts as a client to the MBS front-end.

```
{
    gSystem->Load("/usr/local/marabou_$CERN_LEVEL/lib/libTMrbUtils.so");    // libraries containing class defs
    gSystem->Load("/usr/local/marabou_$CERN_LEVEL/lib/libTMrbConfig.so");   // and methods
    exa = new TMrbConfig("exa", "Example configuration");                   // create a C++ object
                                                                            // to store config data
    rdo = new TMrbEvent_10_1(1, "readout", "readout of camac data");   // define an event (trigger 1) for readout
    cdata = new TMrbSubevent_10_11("data", "camac data");              // define a subevent type [10,11]:
                                                                       // data stored in a zero-padded list
    adc1 = new TMrbSilena_4418V("adc1", "C1.N3");                      // define camac modules
    adc2 = new TMrbSilena_4418V("adc2", "C1.N5");                      // (module type & position)
    cdata->Use("adc1", "etot de1 de2 de3 de4 veto");                   // assign event parameters
    cdata->Use("adc2", "pos1r pos1l pos2r pos2l");                     // to camac modules
    rdo->HasSubevent("cdata");                                         // connect subevent to event/trigger
    exa->WriteTimeStamp();                                             // include a time stamp to raw data
    exa->DefineVariables("I", 1000, "v1[3] v2[4] v3[5]");              // define variables and windows
    exa->DefineVariables("F", 1.2345, "v4[2] v5 v6");                  // (can be modified interactively)
    exa->DefineWindows("I", 0, 1023, "w1 w3 w2");
    sca1 = new TMrbUct_8904("sca1", "C1.N14");                         // some scalers
    sca2 = new TMrbUct_8904("sca2", "C1.N15");
    exa->MakeReadoutCode("", "Overwrite");       // generate C readout code for MBS
    exa->MakeAnalyzeCode();                      // generate C++ class defs & methods for data analysis within ROOT
    exa->WriteToFile("", "Overwrite:Verbose");   // save configuration to root file
    gSystem->Exit(0);
}
```

Figure 3: Example of a configuration macro

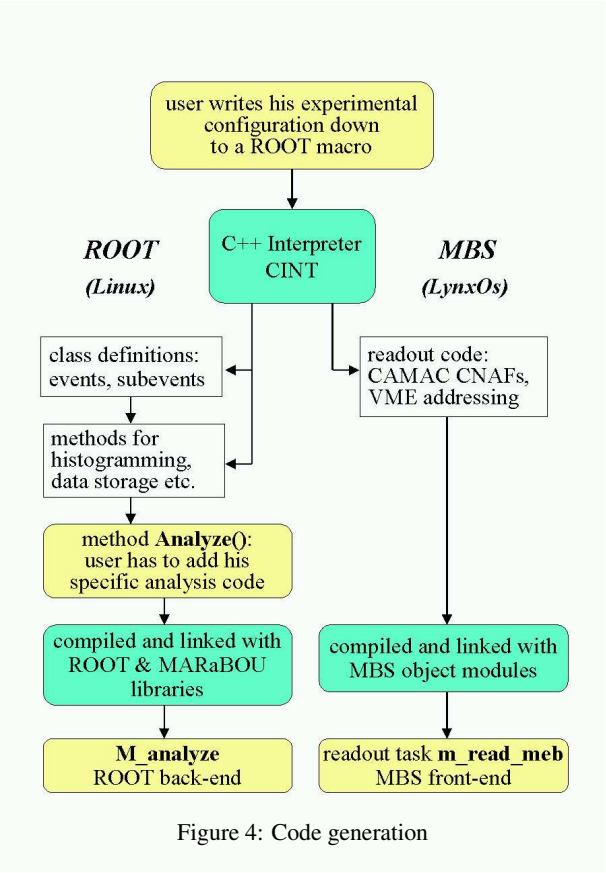


Figure 4: Code generation

IV. Taking Data with MARaBOU

M_analyze contains all the code needed to run the experiment. It is responsible for

- input of MBS data via TCP/IP
- conversion of MBS events to ROOT objects
- histogramming and monitoring
- event filtering
- storage of event data in ROOT format

As M_analyze is able to re-read ROOT-formatted event data from file instead of taking data directly from MBS it may be used without modifications to analyze events in an offline session.

M_analyze is controlled by a graphical user interface (GUI) called C_analyze (fig. 6): it allows to choose the mode of operation (input from TCP or file), to define file names and run ids, to set up the MBS front-end (configure, clear), to control the data flow (start, pause, stop), and to save the accumulated histograms.

Histograms, filter conditions, and fitting functions are stored in a memory-mapped file and thus may be accessed online by other programs.

Fig.5 shows the interaction between M_analyze as a data producer and different consumers. At the end of each run the mapped file is converted to standard ROOT format for later use in an offline session.

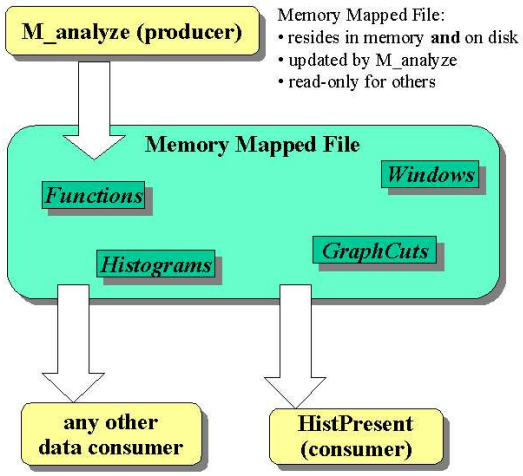


Figure 5: Data exchange via memory-mapped file

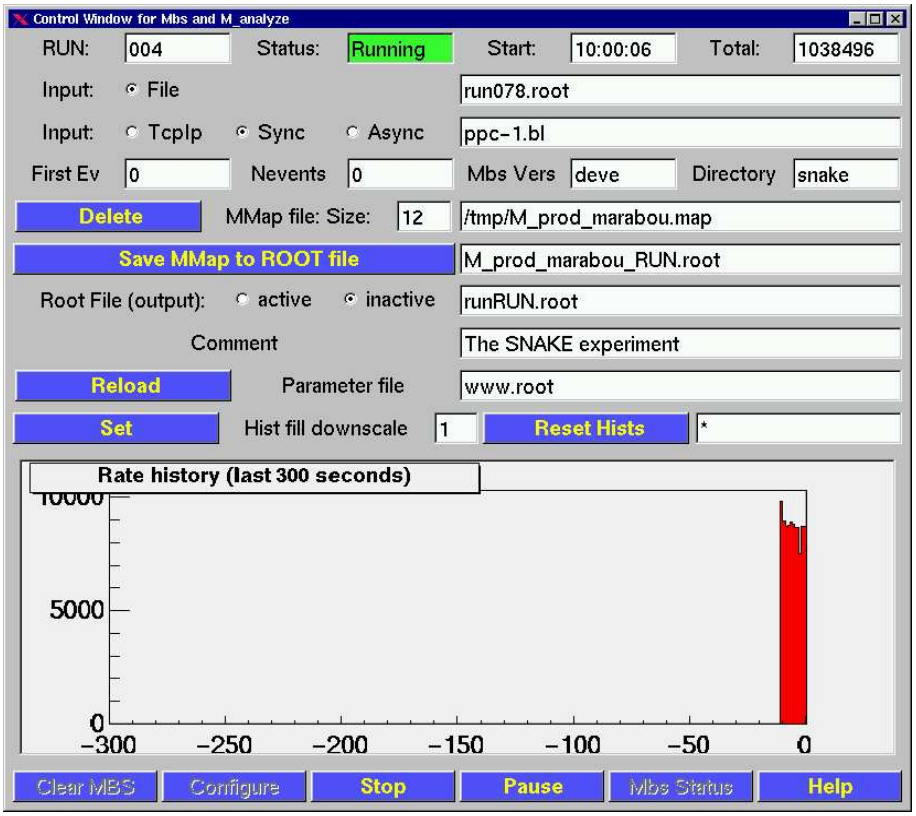


Figure 6: C_analyze: run control

V. HistPresent - A Histogram Presenter

The program HistPresent (fig. 7) provides tools typically needed in an online monitoring environment of nuclear physics experiments. Nonetheless it turned out to be also useful for offline data analysis and presentation.

The program gives simple access to ROOT objects stored in memory-mapped areas or normal ROOT files. The objects handled are 1- 2- or 3-dimensional histograms, 1- or 2-dimensional windows, and functions. It allows operations on histograms like adding, subtracting, multiplying, dividing, scaling. Histograms can be re-binned, zoomed, overlaid and projected. All the graphics options provided by ROOT like lego plots, surfaces and contour plots are available.

Windows used as cuts in 1- or 2-dimensional histograms are important in the analysis. HistPresent allows definition of windows either graphically with the help of the mouse or by typing the values on the keyboard or by a combination of both. They can be stored on files e.g. to be used in an online monitoring program. It might be worth to mention that all ROOT objects on a file carry unique names/versions. This largely simplifies the access of objects and readable programming.

ROOT contains the minimization and error analysis program MINUIT. HistPresent provides a simple interface to the functions of MINUIT. Fitting of Gaussians (optionally with tails) and polynomials is possible just with the help of the mouse. More complicated formula functions may be provided by the user in form of a ROOT macro (essentially C++ code). A built-in template macro makes this an easy task even for a non-experienced user. Functions may be stored to and retrieved from a ROOT file like windows. This is typically used to apply corrections to raw data in an analysis program.

A rudimentary peak finder useful for gamma ray experiments is provided.

All graphical objects can be stored as PostScript or gif files or directly sent to a printer.

Many parameters used by the program like canvas positions and sizes, default file names, printer names etc. may be customized in a resource file (.rootrc). Some parameters like expansion factors, lin-log scale are remembered in and between sessions for each histogram individually.

HistPresent has been developed along with its use in experiments. It therefore mainly reflects the needs of the current users. Thanks to the modularity of ROOT and its built-in C++ interpreter CINT implementation of a new functionality is a simple task.

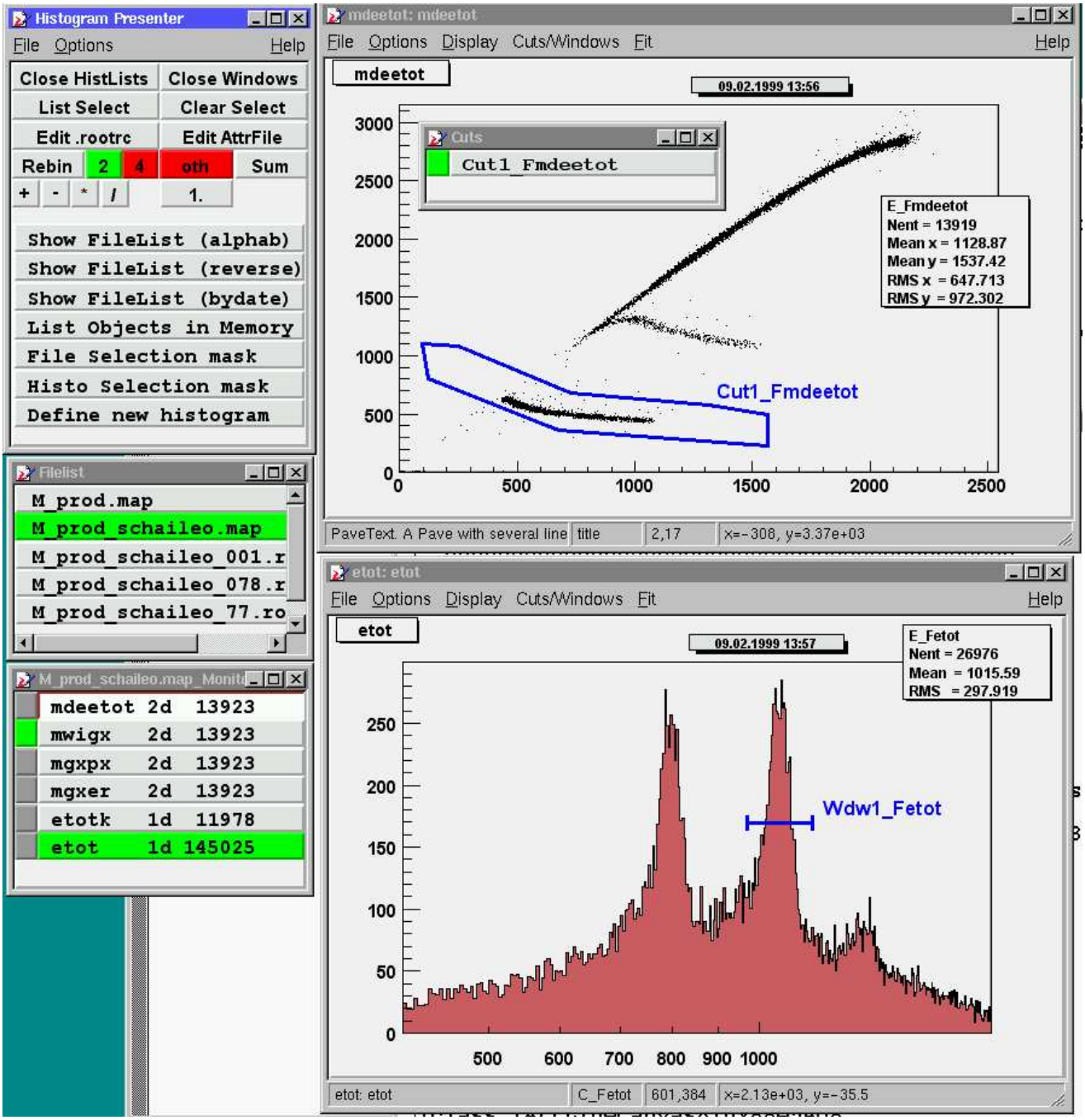


Figure 7: The histogram presenter

VI. References

[1] see also http://www.bl.physik.uni-muenchen.de/marabou/html
and http://www.bl.physik.uni-muenchen.de/marabou/rt99

[2] H. Essel, N. Kurz et al., "The GSI Multi Branch System", http://www-wnt.gsi.de/daq

[3] N. Kurz et al., "Multibranch Data Acquisition System MBS", in *Proceedings of the 11th IEEE Real Time Conference*, Santa Fe, NM, June 14-18, 1999

[4] R. Brun, F. Rademakers et al., "The ROOT Object Oriented Framework", http://root.cern.ch

[5] M. Goto, "The CINT C/C++ Interpreter", http://root.cern.ch/root/CINT.html