Status of ROOT Based Analysis System Go4

J. Adamczewski, H. G. Essel, H. Göringer, M. Hemberger, N. Kurz, M. Richter
GSI Darmstadt

User Requirements

After the decision to design and build a new general purpose analysis system for GSI specific experiments, user requirements have been defined and can be summarized as follows:

▫ *Constraints*: The software shall be based on existing packages (common in the community), platform independent, object oriented and license free.

▫ *User Interface*: It shall provide command, graphical and programming interface.

▫ *Data Management*: Data objects shall be persistent. Objects like histograms shall be exchangeable with other systems.

▫ *Analysis*: It shall support several input sources and implicit or explicit event loops. It shall support large multi-dimensional histograms, provide statistical analysis tools and specific features needed in atomic physics. The controlling interface shall run in parallel to the event loop.

▫ *Graphics*: The GUI shall run in parallel to the event loop, handle complex pictures and produce ready to publish figures. Graphics output from the event loop shall be possible.

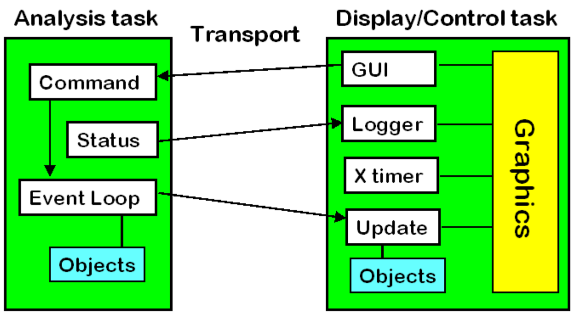
Many requirements, especially the constraints, are met by ROOT[1]. Therefore ROOT has been selected as framework for Go4, the **G**SI **o**bject-**o**riented **o**n-line **o**ff-line system.

In the first phase, mechanisms have been investigated to meet three main requirements: to control a running event loop, to inspect data during a running event loop, and to output graphics from the event loop, e.g. for scatter plots.

Go4 Run Control Structure

There are two structures to implement parallel running tasks: the first is to run separate tasks sharing data in memory and control them through inter-task communication; the second is to run threads in one task, i.e. in the same address space.

Unfortunately, both methods suffer from basic problems: objects cannot be allocated in general in shared memory, and the ROOT classes are not thread save in general.



Go4 Run Control Structure.

Therefore a structure has been designed running two tasks, the analysis and the GUI, without shared memory as shown in the figure. Communication threads in both tasks send commands from the GUI to the analysis and objects from the analysis to the GUI for visualization.

ROOT and Threads

The ROOT thread classes have been updated and are included in ROOT since version 2.22. The main problem is that the `'new'` operator is not thread save. Therefore only one thread may create new objects. In Go4 this is the analysis thread. The communication threads work without object creation.

Concurrent Graphics

Fortunately, the access to canvases is locked in a thread save way, i.e. the canvas pointer is replaced by a pointer function locking the canvas and returning the pointer. Therefore it is possible for a thread in the GUI task, e.g. *Logger* and *Update* in the figure, to paint on the canvas in parallel to the GUI.

Commands are sent from the GUI to the *Command* thread and executed in the *Event Loop* thread. If an object has to be visualized, it is sent from the *Event Loop* to the *Update* thread which does the graphics.

Inter-task Communication

The inter-task communication is implemented using a design pattern called *Bridge*[2]. It separates the interface from the implementation and supports several transport layers. Currently raw TCP sockets, ROOT TMessage and TMapfile have been investigated. We measured that sockets and TMessage are much faster (nearly memory speed) than TMapfile.

Event Handling

A set of hierarchical classes has been designed and implemented to handle several event sources, parse event structures, set up conditions, and connect data fields with histograms. New events can be generated and output to several channels, e.g. root files or standard GSI raw data format.

Event Loop

Since the event loop runs in a thread it can be stopped/canceled and started from the GUI. It executes the commands between the event processing to avoid undefined states. Commands can be ROOT macros providing the full power of prototyping. Designing the event analysis the user can use the dynamic features of the event classes or fix the analysis in the code.

Functional Prototype

After many basic tests a prototype of the run control mechanism has been implemented on Linux to proof the principle. Through a simple GUI one can start/stop the analysis, switch the event input source on the fly, create objects like histograms in the analysis task, display them, set up histogram filling conditions, and display scatter plots. Event rates are calculated in the *Status* thread, sent to the *Logger* and displayed.

There are still stability problems to be investigated, caused by the combination of threads, ROOT and Linux. On SMP machines threads are not at all running stable due to problems in the Linux Xlib. We hope this bug will be fixed in future Linux releases. The actual status can be found on the Go4 Web site http://www-wnt.gsi.de/go4.

[1] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86, http://root.cern.ch

[2] Erich Gamma et al., *Design Patterns: elements of reusable object-oriented software*, Addison-Wesley Professional Computing Series, ISBN 0-201-63361-2