Go4 On-line Monitoring

J. Adamczewski, M. Al-Turany, D. Bertini, H.G. Essel, N. Kurz, S. Linev, M. Richter

Abstract-- **Go4 developed at GSI is an analysis framework with a general purpose non blocking GUI. Go4 is based on ROOT. The GUI is implemented in Qt using GSI's QtROOT interface. Analysis and GUI run in separate tasks communicating through asynchronous object channels. Therefore the framework is especially useful for on-line monitoring where an analysis should run continuously, but controlled at any time from the GUI. The analysis also should be able to update graphical objects in the display. The HADES experiment at GSI integrated an existing ROOT based monitoring analysis into Go4.**

A Go4 analysis may use any ROOT features. It can be organized in steps, which can be controlled from the GUI according to the user specifications. Each step is composed of event objects, the event processing, and event IO.

Go4 composite event classes allow the construction of arbitrary complex events hierarchically composed of objects. The IO of the composite event objects to and from ROOT trees/branches is provided without explicit programming. Arbitrary hierarchy levels of composite events can be browsed by the Go4 tree viewer.

The GUI provides hooks to attach user written GUIs. These GUIs have access to all objects of the analysis, i.e. events for asynchronous event display. Using the Qt designer the development of such GUIs is very efficient. The HADES experiment implemented a dedicated GUI for the on-line monitoring.

The Go4 fit package (API and GUI) is a powerful and extendable tool to model and fit experimental data.

I. Overview

The GSI on-line off-line object oriented analysis framework Go4 is based on ROOT [1]. The Go4 Graphical User Interface (GUI) is implemented using the Qt graphics library [2]. It interoperates with ROOT through the QtRoot interface developed at GSI [3].

The main features of Go4 are:

- It provides a framework for atomic and nuclear physics experiments.
- The analysis is written by the user (unlimited ROOT).
- The GUI controls and steers the analysis.
- The GUI is never blocked by a running analysis. The analysis is never blocked by the GUI. Both features are important for on-line use.
- The analysis may update graphics in the GUI asynchronously.
- The GUI can be extended by user written GUIs.
- The analysis may run without modifications in batch mode.

Manuscript received June 6, 2003; revised October 10, 2003.
J.Adamczewski@gsi.de, M.Al-Turany@gsi.de, D.Bertini@gsi.de, H.Essel@gsi.de, N.Kurz@gsi.de, S.Linev@gsi.de, M.Richter@gsi.de
Gesellschaft für Schwerionenforschung, GSI
64291 Darmstadt, Planckstr. 1, Germany.

II. GUI Task and Analysis Task

A. Inter-task Communication

To control an on-line or off-line analysis without blocking from the GUI, analysis and GUI run in separate tasks connected via sockets and communication threads [4]. Fig. 1 shows a schematic view of the tasks with their functional components. A complete object transfer between both tasks is provided by means of the ROOT streamer mechanism. Exchange of command, data, and status objects allow control of the analysis task from the GUI. Any ROOT object from the analysis side can be requested from the GUI side. Both, the Go4 remote object browser, and user written GUIs may request and receive analysis objects. Such objects also can be sent asynchronously by the analysis. The optional histogram client/server connection is described below.

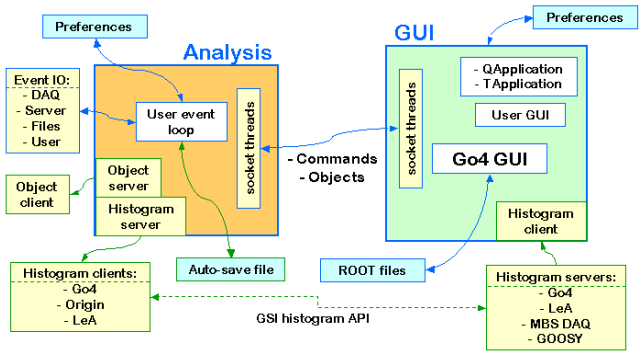

Fig. 1. Analysis and GUI tasks

B. The GUI Registry

The dispatching of the incoming objects on the GUI side is handled by the GUI registry. GUI commands that request an object from the analysis register here which action shall happen when the object arrives, e.g. drawing a histogram, or passing a parameter object to an editor window. The registered action will be executed on arrival synchronized with other GUI activities. Similarly, a user GUI can register names of objects to be handled by itself. By this the user GUI can handle user objects sent by the analysis without a corresponding request from the GUI (e.g. on-line event information that are only sent when an "interesting" event occurs).

C. Histogram Client/Server

Go4 allows the histograms of a single analysis to be accessed by many clients, e.g. Go4 GUIs. This is achieved by an optional histogram server/client mechanism shown in Fig. 1.

This mechanism uses a standard GSI histogram API library written in C (available from the Go4 web site as part of the GSI event API library). The API provides functions to build histogram servers and clients. At GSI, servers are implemented in the Go4 analysis, the standard GSI data acquisition MBS [5], and the GSI analysis frameworks LeA [6] and GOOSY. Clients are available in the Go4 GUI, LeA, and as add-on in Origin of OriginLabs [7]. Other clients can be easily implemented using the API. Any client can access histograms from any server in an IP network. It might be especially useful in Origin to have direct access to histograms of a running Go4 analysis.

In the Go4 GUI the histograms (having their own format on the different servers) are converted to ROOT histograms. They can be modified and saved into ROOT files.

D. *Object Server*

Since the GSI histogram API is dedicated for histograms only, it can not transport other ROOT objects, e.g. conditions, parameters, or user defined ones. Therefore Go4 offers an additional object server running in the analysis. It uses a ROOT streamer and socket mechanism similar to the inter-task connection between analysis and controlling GUI (Fig. 1). A user written ROOT application may apply the corresponding object client class to request any analysis object from this server.

III. Analysis Framework

The Go4 analysis framework can run both in batch mode or in GUI controlled multi-threaded mode. Many kinds of user analysis codes can be implemented using the framework, i.e. Go4 classes and interfaces.

A. *Base Classes*

User analysis and event classes inherit from a set of Go4 base classes. These define interfaces for the framework to handle any analysis in a common manner. Base classes exist for event structures, event processors (algorithms), event IO (source and store), and the analysis frame itself. Classes for standard GSI event sources and several types of events are provided. The implementation of other event sources is supported by examples. Event objects are stored/retrieved in ROOT trees.

B. *Analysis Steps*

A Go4 analysis is organized in steps. Each step has an event source, an input event structure, an event processor, an output event structure, and an event store. Fig. 2 shows an example with two steps: "Unpack" and "Calibrate". These are implemented by the user. The steps are set up at initialization time by a factory instance for each step.

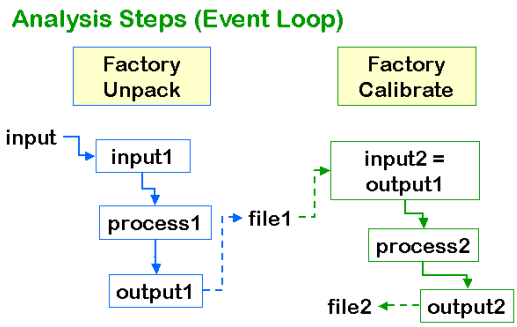

Fig. 2. Analysis steps data flow.

Analysis steps run sequentially, each working on the output event of the previous step. Moreover, each step can work independently, getting input from its own event source (file), and writing output into its own event store (file) (Fig. 2). The execution of the steps and the event IO is set up in the factory (e.g. used in batch mode) and can optionally be controlled from the GUI.

C. *The Go4 Composite Event.*

The composite event classes can be used to build complex hierarchically structured event objects reflecting the natural experimental setup. Complex events are built out of simple data objects and/or composition of data objects uniformly.

Fast direct access of components is realized by indexing. Full or Partial IO is achieved by mapping the event object into a ROOT TTree. Composite events are stored using standard ROOT IO mechanisms without changes in TTree or TBranchElement. Event store/retrieval, fully or partially, is done in the base classes by recursive mechanisms without the need of extra user written code. The Go4 tree browser resolves the tree hierarchy up to unlimited levels.

IV. GUI Controlling the Analysis

A. *Analysis Control*

A Go4 analysis can run in batch mode, command line mode, or in the ROOT CINT interpreter. It also can be started from the GUI. In this case the complete analysis setup can be controlled from the GUI. The analysis configuration window in the GUI is formatted according to the analysis steps defined by the user. In Fig. 3 at the right side one sees the configuration window for the example of Fig. 2 with two pads for "Unpack" and "Calibrate", respectively. Each step can be (de-)activated, and the event IO can be specified.

In the configuration window one also can write all registered objects like histograms, parameters, conditions into the auto-save file. A configuration file can be saved/restored.

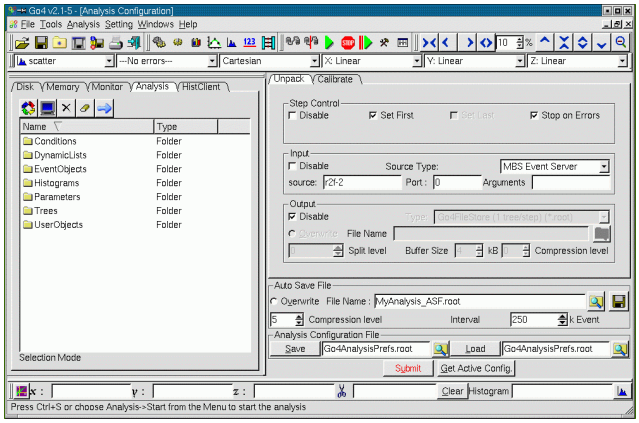

Fig. 3. Analysis configuration panel.

B. *Remote Object Browser*

At the left side of Fig. 3 one sees the browser pads. Any ROOT object located in the analysis can be viewed and requested from the GUI via a remote object browser (Analysis pad). For an object to be visible in the browser the analysis must enter it into (sub-) folders of the framework. Objects like histograms can be copied into the local memory (controlled by the Memory pad) or selected for monitoring (controlled by the Monitor pad). They can be displayed in a Go4 view panel. Other objects like parameters can be fetched by double click to be edited in the corresponding editor window. In Fig. 3 the top folders of the analysis are shown at the left side. In Fig. 4 some folders are opened. Two other pads are provided for file browsing (Disk) and histogram server access (HistClient), respectively.

C. *Condition Editor*

The Go4 analysis conditions are a set of classes dedicated to test values against given boundaries (windows or polygon shapes). These can be visualized and edited from the GUI. Fig. 4 shows an example of polygon conditions.

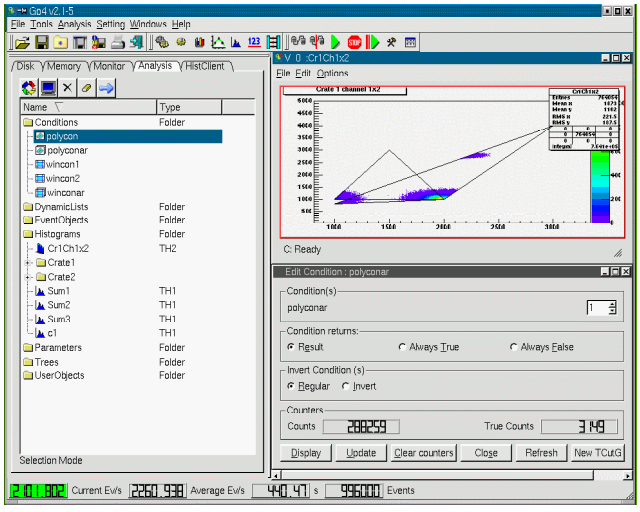

Fig. 4. Condition editor.

All conditions can be frozen to be always true or false, respectively. Using this feature one can use conditions to steer the analysis by executing parts of the code only when a condition is true. The test and true counters provide useful information about the statistics of the condition results.

D. *Generic Parameter Editor*

The TGo4Parameter base class offers a mechanism to edit user defined structures of values on the GUI and apply them in the analysis for any purpose. The user parameter subclass may contain members of any basic data type or arrays of these.

The member declarations are evaluated for display in the generic parameter editor, using the ROOT class information. An example is shown in Fig 5. New values can be entered. Pressing "Apply" the parameter object will be sent to the analysis.

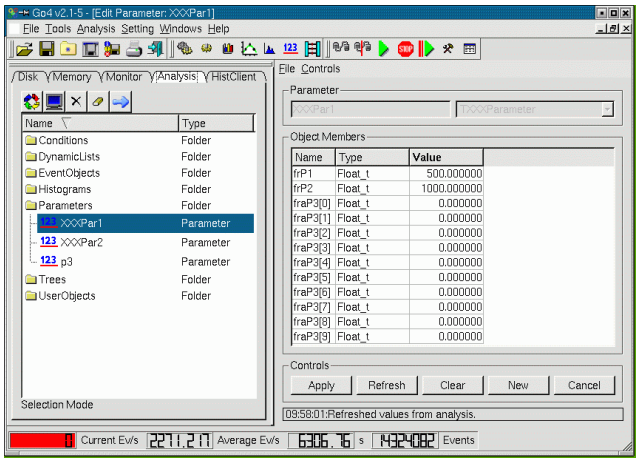

Fig. 5. Parameter editor.

E. *Histogramming on the Fly*

The Go4 dynamic list is a mechanism to define connections between histograms, conditions, and event data values on the fly without recompiling the user analysis. The histograms and conditions can be created interactively. Two alternative approaches are implemented:

1. using the standard ROOT TTree::Draw mechanism. Tree output must be enabled in the analysis configuration. Then the events stored in that tree are processed in regular intervals. When the tree is processed it is reset. Composite events are supported.

2. by direct pointer access to the event object members using the ROOT dictionary meta information. Only events containing basic data type members are supported. The histograms are filled event by event.

Both variants can be controlled from the GUI.

V. Go4 Fitter

The Go4 fitter package is an independent add-on to ROOT [8]. It features a modular extensible design capable of using any kind of minimizers, fit functions, and models, inside and outside ROOT. The Go4 GUI offers a fitter window to per-

form fits on any 1 and 2 dimensional histograms in memory or in a ROOT file (Fig. 6).

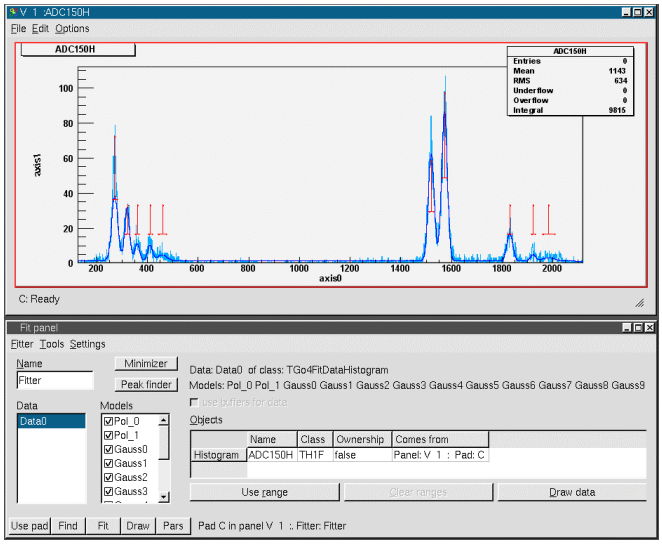

Fig. 6. Fit panel with example fit.

Automatic peak finders allow a fast setup of fit models and initial parameters for a given histogram. All peaks in Fig. 6 have been set up by the peak finder. The models can be modified if necessary. Fit and results can be controlled both in an intuitive wizard mode, and in an expert mode with full access to entire functionality of the Go4 Fit API.

VI. Status

The Go4 framework has been developed to replace all outdated analysis programs at GSI, especially GOOSY based ones. Many experiments at GSI like the Fragment Separator (rare isotopes) [9] or SHIP (heavy elements) [10] have already implemented analysis programs using Go4. Currently these programs run for testing on-line in parallel to the well proven GOOSY based ones.

The new RISING [11] experiment is using Go4 for on-line (several recent beam shifts) and off-line analysis.

The HADES collaboration [12] uses Go4 based programs on-line. Their existing ROOT based analysis programs have been embedded into the Go4 framework. Go4 offers hooks to attach a user written GUI to the main Go4 window, with the possibility of full interaction with the analysis, i.e. sending commands, requesting analysis objects, etc. The HADES user GUI makes heavy use of these features. It has been developed with Trolltech's Qt designer tool delivered with Qt [2]. Fig. 7 shows a screenshot. The information in the graphics window is updated from the analysis continuously. The buttons send user commands to the analysis for execution.

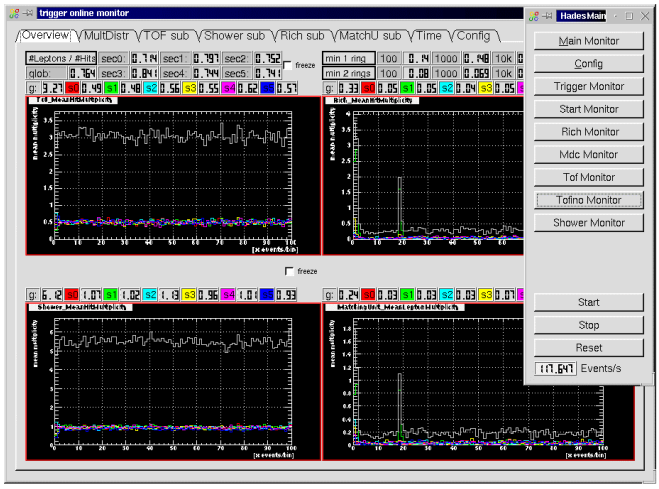

Fig. 7. HADES online display user GUI.

VII. Conclusion

Go4 is a versatile framework that has already proved to fulfill most of the requirements for the GSI medium sized experiments. It is still under continuous improvement. The standalone Go4 GUI can be used as an interactive analysis tool for all ROOT based data.

Many existing event loop styled ROOT analysis programs could be adopted to run inside the Go4 framework, with the benefit of non-blocking GUI control from a remote task. New analysis programs can make use of the Go4 analysis steps logic, and may run both in batch or GUI mode. User defined GUIs can be added with full access to all analysis objects.

Go4 v2 is now available for download from the web site http://go4.gsi.de/. It is tested on Linux (Debian 2.2, Debian 3.0, RedHat 7.2, Suse 8.0) with several compilers (gcc 2.9, gcc 3.2, icc 7.0).

References

[1] R.Brun and F.Rademakers, "ROOT – An object oriented Data Analysis Framework", *Nucl. Inst. Method Phys. Res.*, vol.A389, pp. 81-86, 1997

[2] Qt 3.1 Tutorial and Reference, Troll Tech AS, http://www.trolltech.com/

[3] The QtRoot interface, http://www-linux.gsi.de/~go4/qtroot/html/qtroot.html

[4] J.Adamczewski, M.Al-Turany, D.Bertini, H.G.Essel, M.Hemberger, N.Kurz, *et al.*: "Go4 multitasking Class Library with ROOT", *IEEE Trans.Nucl.Sci*, Vol.49, No 2, pp. 521-524, April 2001

[5] MBS, http://daq.gsi.de/

[6] LeA: http://lea.gsi.de/

[7] OriginLab, http://www.originlab.com/www/products/origin

[8] J.Adamczewski, M.Al-Turany, D.Bertini, H.G.Essel, S. Linev: The Go4 Analysis Framework: Fit Tutorial, http://go4.gsi.de/Docs/Go4FitTutorial.pdf

[9] FRS: http://www-wnt.gsi.de/frs

[10] SHIP: http://www.gsi.de/ship

[11] RISING: http://www-linux.gsi.de/~gsgweb/index.html

[12] J. Markert, J.Adamczewski, M.Al-Turany, D.Bertini, T.Christ, A.Gabriel, *et al.*, "HADES online-monitoring with Go4", GSI Scientific Report 2002, p. 215.