DISTRIBUTED OBJECT MONITORING FOR ROOT ANALYSES WITH GO4 V3

J. Adamczewski, H.G. Essel, S. Linev; GSI, Darmstadt, Germany

Abstract

The new version 3 of the ROOT based GSI standard analysis framework Go4 (GSI Object Oriented Online Offline) has been released. Go4 provides multithreaded remote communication between analysis process and GUI process, a dynamically configurable analysis framework, and a Qt based GUI with embedded ROOT graphics. In version 3 a new internal object manager was developed. Its functionality was separated from the GUI implementation. This improves the Go4 GUI and browser functionality. Browsing and object monitoring from various local and remote data sources is provided by a user-transparent proxy architecture. The Go4 communication mechanism between GUI and analysis processes was redesigned. Several distributed viewers may now connect to one analysis. Even a standard CINT session may initiate the Go4 communication environment to control an analysis process with the native ROOT browser. Similarly, standard analysis ROOT macros may be controlled by either a remote Go4 GUI or ROOT browser. Besides Linux, a lightweight binary Go4 v3 distribution (without Qt GUI) for MS WindowsXP is now available. Cross platform connections between Go4 environments are possible.

GO4 INTRODUCTION

The Go4 (GSI Object Oriented Online Offline) software package is well established in the atomic and nuclear-structure physics community at GSI. It is based on the ROOT system [1]. Go4 v3 was released in November 2005 and can be downloaded from the Go4 website [2] under GPL conditions.

The Go4 analysis framework applies flexible interface classes for data structures, IO, event processing, and runtime initialization. The analysis can be set up modularly by means of "analysis step" objects. The experiment specific analysis code is written in user-defined subclasses and virtual methods. All ROOT and C++ features may be applied here, thus it is also possible to adapt code from other frameworks. Implementations for the GSI standard DAQ system MBS [3], and ROOT TTree IO are already provided.

The same Go4 analysis may either run in batch mode (compiled or as CINT macro), or in an interactive mode controlled by a non blocking GUI. Here analysis and GUI run in separate processes, connected for asynchronous command and data exchange via sockets. This multi threaded inter-task communication is managed by dedicated Go4 libraries [4].

The Go4 GUI was designed by means of the Qt graphics library [5]. It implements regular ROOT

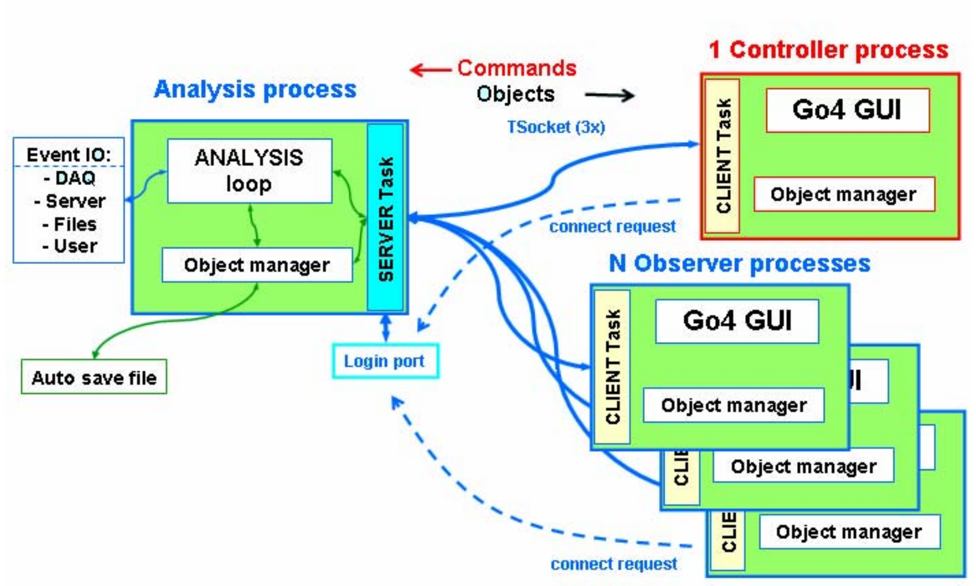


Figure 1: Go4 inter-task communication: one analysis server (slave) is connected from several GUI clients (masters). There is only one controller GUI allowed that may modify the analysis set up and running state.
]

graphics inside Qt widgets [6]. A multi document interface (MDI) workspace offers subwindows for object browsing, histogram display, editing, analysis configuration, etc. Moreover, it is possible to extend the standard GUI functionality by experiment specific windows, as defined by the user with the Qt designer tool.

To enhance functionality and maintainability, some internal components have been redesigned for Go4 version 3. This mainly concerns the inter-task communication, and the object management of the GUI.

NEW DEVELOPMENTS FOR V3.0

Inter-task Communication

The design upgrade of the Go4TaskHandler architecture [4] resulted in a decoupling of the client/server functionality from master/slave roles. Thus the analysis can run in the network both as server or client task. Vice versa, the GUI can run either as client or as server. As a benefit, one analysis server can be connected by many Go4 GUIs (Fig.1). A crash of one GUI will simply remove the connection at the server, without affecting the analysis. Moreover, all GUIs may even disconnect from the running analysis completely without shutting it down.

A simple account management prevents conflicts of control between the different GUIs. For login of the GUI at the analysis task, there are observer, controller, and administrator accounts. The **observer** may only request exisiting data and display it, but can not modify data, or change the analysis set-up or running state. The **controller** may in addition start and stop the run, and change the analysis parameters. Finally, the **administrator** may even shut down the analysis server completely and thus disconnect all other GUIs.

There can be only one controller (or adminstrator) connected to the analysis at the same time. The number of observers is not restricted in principle, but may practically be limited by the performance of the server task. A test was done with 15 GUIs at one analysis, distributed over different nodes, with many permanent monitoring requests from all observers. The analysis server showed to run stable until it was stopped after 3 days. However, analysis event rate dropped by a factor of 3-5 compared to the case with only one GUI. This of course strongly depends on the kind of analysis, and on the number and frequency of requested objects.

The changed task handler design made it also possible to use a Go4 analysis server in a regular ROOT CINT session. This is described in more detail in section “distributed monitoring”.

Object Manager

The new object manager (OM) is a general registry for data from different sources, using full pathname identifiers. It defines a common API also suitable for interactive work.

The OM keeps a hierarchical structure of containers, with proxy objects for transparent access to different data sources (see Fig.2). There are proxy implemenations for ROOT TFile, TDirectory, TFolder, TTree, TCanvas, respectively; the socket connection to the remote Go4 analysis task is also wrapped inside a proxy class.

The OM applies the standard ROOT cleanup mechanism for consistency of data references. Additionally, if a data element is changed, a message passing mechanism between different OM branches ensures the notification of dependent elements, e.g. to refresh a corresponding view.

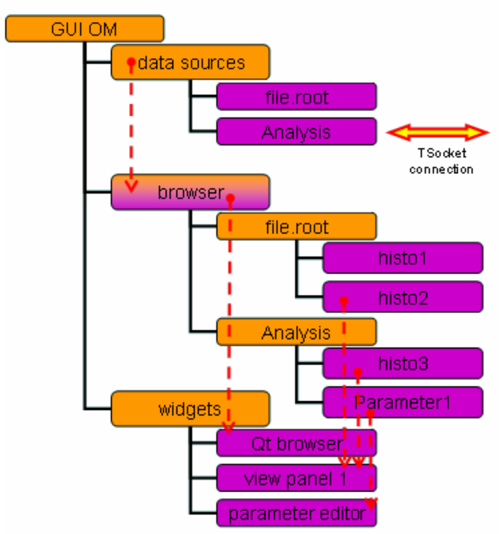


Figure 2: Object manager for GUI using containers (orange), proxy objects (violet), and message passing in between (dashed arrows).

In the Go4 v3 GUI, the OM handles all data objects, views, editors, and browser widgets. However, the object management is completely decoupled from the views. Thus an alternative Go4 GUI can be implemented using other widget libraries than Qt. As an example, Go4 v3 offers a lightweight master task environment that uses a CINT session with ROOT graphics (see section “distributed monitoring”).

GUI Enhancements

As a consequence of the OM redesign, many elements of the Qt GUI were improved.

The **object browser** was completely re-implemented as one browser panel with branches for different data sources, e.g. remote analysis, histogram servers, root files, etc. It supports a local memory workspace with copy and paste, drag and drop, clipboard, and renaming functionalities. Other features are context menus, a general monitoring management, and a property filter. The internals of the renewed **editor and property windows** (for conditions, parameter structures, dynamic histogramming settings, etc.) are also fully driven by the OM.

Simlilar changes were done for the **viewpanels** that embed ROOT TCanvas graphics in the Qt MDI environment. Because the OM separates data and view, the same object may be displayed multiple times with different draw options and ranges simultaneously.

A new **mbs status widget** monitors an MBS data acquisition node [3] directly. Trending histograms of event rates vs. time may be created in the local workspace of the Go4 browser.

DISTRIBUTED MONITORING

Go4 GUI remote features

The separation of analysis (slave) and GUI (master) into different processes leads to a non-blocking GUI that may control the slave asynchronously. Moreover, since the inter-task communication is done by TCP sockets, both tasks may also run on different nodes. The task handler improvements, as described above, make it even possible to use several observer GUIs for monitoring of one analysis slave, all arbitrarily distributed over the network.

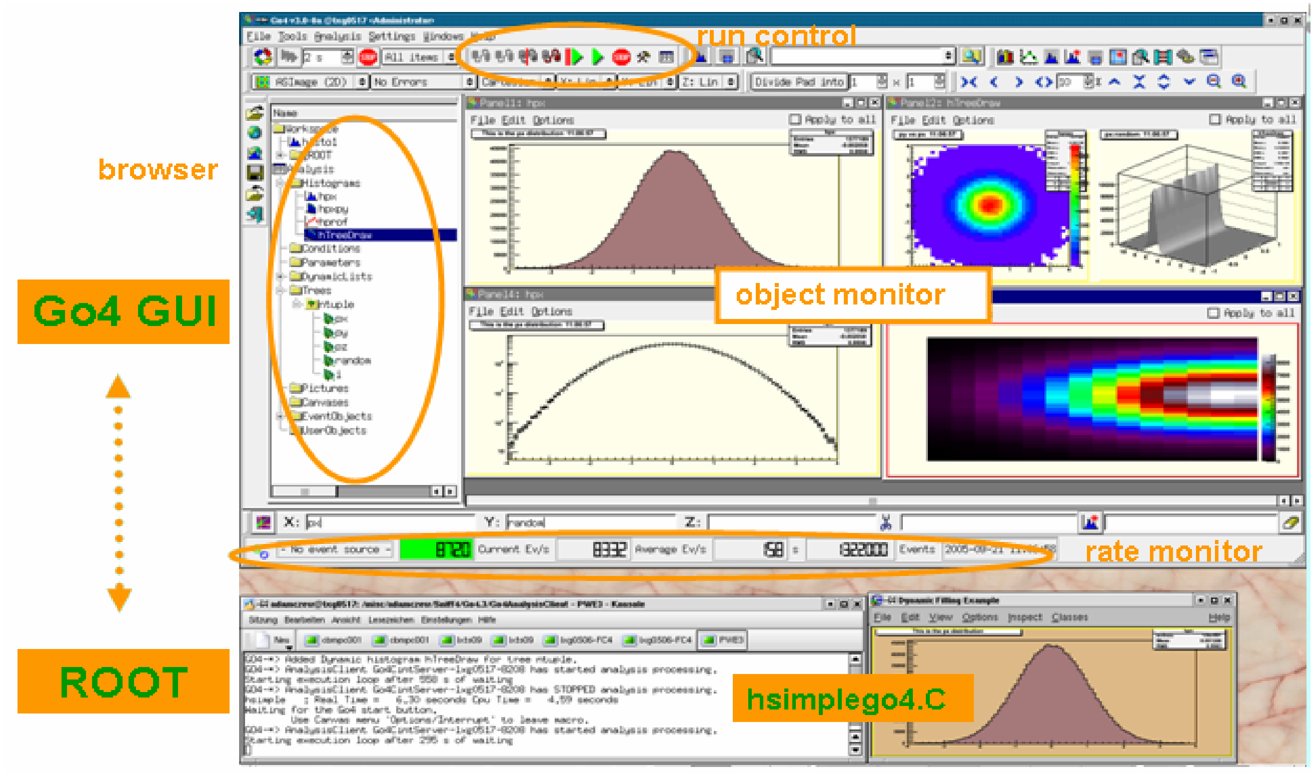
The Go4 GUI offers by default some interesting features for remote control and object monitoring [7]. The objects on the slave side can be registered to the framework in a hierarchical folder structure, sorted by type (histograms, conditions, parameters, event data, etc.). This structure is available from remote by means of the Go4 GUI browser. The contained objects may be inspected by properties and drawn into a view panel. The fetching of object copies from the slave is done in a user transparent way, i.e. working on remote objects has the same “look and feel” as for local memory or file objects. Besides, Go4 provides a “monitoring” functionality: objects may be selected in the browser to be fetched automatically from the slave in frequent time intervals; their graphical representation will be refreshed on the GUI. These features are available for all observers.

If the GUI is connected with controller priviliges, it may additionally modify, create or delete objects in the remote process. Furthermore, a controller GUI may schedule any ROOT macro or CINT command to be executed in the remote process.

It should be pointed out that the above features are independent of the Go4 analysis framework itself, but use the Go4 communication layer and object registry only. If in addition the slave process applies the Go4 analysis loop and the event classes, the controler may also initialize the analysis, start/stop the loop run, and get event rate information. New histograms may be set up in the remote analysis “on the fly” to be filled from event data in memory, or from a ROOT TTree, respectively.

Go4 GUI controlling remote ROOT sessions

The Go4 analysis server environment was extended to be run in a regular ROOT CINT session. The provided initialization script loads the Go4 libraries and launches the task handler threads in the background of the ROOT command line. To ensure thread safety for any user code that might be executed in CINT, a ROOT timer with 200 ms period is used. This timer locks all Go4 mutexes during the processing of an interpreter line, and releases them for a dedicated time interval of 50 ms when the timer is fired. The Go4 threads can use the mutex protected ROOT framework code within this interval



```
GO4-*> Added Dynamic histogram hTreeDraw for tree ntuple.
GO4-*> AnalysisClient Go4CintServer-lxg0517-8208 has started analysis processing.
Starting execution loop after 558 s of waiting
GO4-*> AnalysisClient Go4CintServer-lxg0517-8208 has STOPPED analysis processing.
hsimple   : Real Time =   6.30 seconds Cpu Time =   4.59 seconds
Waiting for the Go4 start button.
      Use Canvas menu 'Options/Interrupt' to leave macro.
GO4-*> AnalysisClient Go4CintServer-lxg0517-8208 has started analysis processing.
Starting execution loop after 295 s of waiting
```

Figure 3: Go4 GUI controlling remote ROOT macro

only, i.e. in between two lines of a macro.
Having started this environment, the ROOT session can be controlled from a Go4 master process (Qt GUI or other root session, see below). All Go4 remote features are available here: the GUI process may inspect and request data from a running analysis macro. If this macro in addition uses some methods of the Go4 API, the GUI can control the running state with the Start/Stop buttons and monitor the event rate. Figure 3 shows the screenshot of a Go4 GUI process connected to a ROOT session in another process. Here a slightly modified *hsimple.C* macro from ROOT tutorials is controlled. Main remote functionalities of the GUI are highlighted in the figure.

ROOT sessions controlling Go4 analysis

The separation of the Go4 GUI object management from the graphics layer makes it possible to replace the standard Qt widgets by any other surface. Hence a first leightweight UI for the Go4 master task was developed that purely uses ROOT graphics. This master environment can be started in a regular CINT session, similar to the analysis server environment as described above. Some extensions of the regular TBrowser provide the Go4 remote browsing and monitoring functionalities, with standard TCanvas graphics for display. In addition, a command line API offers full control over the connected analysis task. This may be used both interactively and in set-up macros.

As a benefit, Go4 can also be installed and run in environments where Qt graphics is not applicable. This is the case for MS Windows XP: although the Qt library is available, the current Go4 Qt-ROOT interface [6] does not work here for principal reasons. However, using the lightweight master UI only, Go4 could be ported successfully to this platform. A binary Go4 distribution for Windows XP is available for download [2].

Some use cases

The new Go4 functionalities may be used for distributed object monitoring in various scenarios:

A compiled Go4 based analysis runs in a slave server task on a linux node and receives online data from a DAQ system. The run is permanently controlled by one administrator GUI at the measurement site. Observer GUIs at the facility may connect to the server and display data of dedicated subdetectors each. Additionally, guest observers may connect at any time from anywhere in the network, e.g. a windows notebook running a pure ROOT observer, or a full featured Go4 GUI on a Linux PC in a university office.

A compiled Go4 analysis is set up from a controller GUI to process offline data from file. After analysis start the controller may disconnect; it may reconnect again later from a different node to inspect results, or to change settings, respectively.

A compiled Go4 analysis server uses an **external ROOT based analysis framework** in the Go4 eventloop. A controller GUI process is permanently connected and may start or stop the run, or modify object contents (e.g. zero histograms). The event rate, and the registered objects may be displayed on any remote observer GUIs.

A ROOT macro with a Go4 analysis server task is processing tree data and has registered all ROOT memory objects in Go4. This analysis job may work without GUI in a pseudo batch mode for a long time. Go4 observer GUIs may connect to this process and display intermediate results.

A ROOT session with a Go4 observer client task runs a macro that connects subsequently to different Go4 analysis server nodes, retrieves result histograms from each, disconnects again and continues with the next analysis node. All results are copied to the local Go4 workspace and may be merged together, or processed further, at the end of the retrieval cycle.

Of course there are many other use cases to imagine here. Practically all combinations of the following are possible:

- roles (UI master, analysis slave)
- tasks (server, client)
- environments (compiled Go4, ROOT CINT session, compiled ROOT)
- analysis jobs (Go4 analysis steps, foreign framework, plain ROOT)
- platforms (Linux, Windows)
- number of nodes and degree of distribution

CONCLUSIONS

The Go4 framework, as established in GSI since 2002, has been further developed to release version 3. The redesign of inter-task communication architecture and a new object management have extended functionality. Many distributed observer tasks may monitor one analysis process via the network. The remotely observed analysis process may also be a regular ROOT session. The Go4 Qt GUI was improved. An alternative user interface for regular ROOT sessions is provided. Because of this, Go4 could be ported without the Qt-ROOT layer to the Windows XP platform.

REFERENCES

[1] http://root.cern.ch
[2] http://go4.gsi.de
[3] http://daq.gsi.de
[4] J.Adamczewski et al., *Go4 multitasking class library with ROOT*, IEEE TNS Vol.49, No.2, April 2002, pp 521-524
[5] http://www.trolltech.com
[6] http://www-linux.gsi.de/~go4/qtroot/html/qtroot.html
[7] J.Adamczewski et al., *Go4 online monitoring*, IEEE TNS Vol.51, No.3, June 2004, pp 565-570