GO4 ANALYSIS DESIGN

J. Adamczewski, M. Al-Turany, D. Bertini, H.G. Essel, S. Linev
Gesellschaft für Schwerionenforschung GSI, Germany

Abstract

The Go4 framework has been developed for Atomic and Nuclear Physics experiments at GSI. The current analysis design as provided by Go4 [1] is described. New advanced requirements are reported. From a linear organization of the analysis one wants to go to a mesh like structured analysis with multiple data (event) streams. These requirements can be met by a two step upgrade. The first step adds some improvements, the second implements a new concept of the analysis organization.

GO4 FRAMEWORK

Go4 is based on the ROOT system. Besides the full ROOT functionality Go4 offers additional services to develop analysis programs. Such programs may run in batch mode, or interactive mode, respectively. Interactively the analysis is controlled by the Go4 GUI.

Analysis and GUI Tasks

Analysis and GUI run in separate tasks communicating through sockets [2]. Therefore the analysis can run permanently without blocking the GUI. It can, however, send data asynchronously to the GUI for display. This feature is especially useful for on-line monitoring [3].

Go4 GUI

The GUI is developed with Qt (ROOT graphics embedded via GSI's QtRoot interface). User written GUIs can be plugged in. The GUI has full control of the analysis. All registered analysis objects can be browsed. Specific types can be edited. The full ROOT graphics is available. User GUIs may be plugged in.

GO4 ANALYSIS

Go4 Analysis Organization in Steps

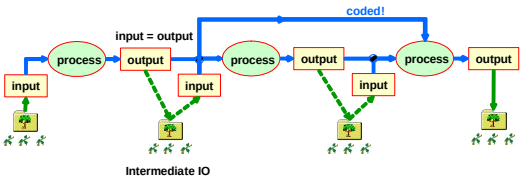

Figure 1: Chain of analysis steps.

The Go4 framework handles event structures, event processing, and event IO. As shown in Fig. 1 the analysis event loop is organized in **steps**: Each step has an **input** event, an **output** event, and an event **processor**. To be filled the output event calls a method of its event processor. The event processor has a reference to the input event. Output events may be written to files. Input events may be retrieved from files or from previous step(s). The information needed to create the event and processor objects (which are deleted when the event loop terminates) is stored in step **factories** which are kept in the **analysis singleton**.

Analysis Control

When the analysis task is started all step, factory, event, and processor objects are created. When started by the GUI the following controlling mechanisms are available:

1. In the step configuration window steps can be dis/enabled. The event IO can be set up.
2. ROOT macros, commonly used to configure an analysis, can be launched from the GUI. They execute in the analysis task synchronized with the event loop (if the event loop is running).
3. User parameter objects retrieved from the analysis task can be edited with a generic object editor in the GUI. When the modified object is sent back to the analysis an *UpdateFrom* method is called. In this method the user has full access to the analysis framework.
4. Conditions (window and polygon) are used in the analysis code to check values against limits or polygons. In the GUI condition editor not only the limits and polygons can be set, but also the behaviour, i.e. condition shall return always true or false. Using "dummy" conditions one can steer the analysis code. The execution and true counters of the conditions give useful hints about program execution.
5. With the dynamic histogramming feature one can create histograms and conditions on the fly and fill them event by event (or from tree) with members of the event objects.

NEW REQUIREMENTS

Event Stacks

Sometimes an analysis needs more than one event to work. E.g. if the events delivered by an asynchronous DAQ are in fact logical sub-events marked by time stamps the analysis has to built the real event from events of a time interval. Another example would be decay experiments when the analysis needs to handle a set of events following an event of a certain signature. Of cause, all this could be handled on application layer. But if the framework provides such a feature it saves developer time.

Step Hieararchy

The current design of the steps is linear and oriented towards data generations. If the experiment utilizes several detector systems, a hierarchical approach like the ROOT TTask mechanism would be more natural. The execution order of such tasks is fixed and top down.

Concurrent Steps

Similarly, dealing with several detectors, it would be more natural that the detector codes are assigned to steps but process the same input and output events which of cause could be complex structures.

Multiple I/O

There might be cases where the events are split into ROOT tree branches Then a step may need event fragments from several branches.

Analysis Mesh

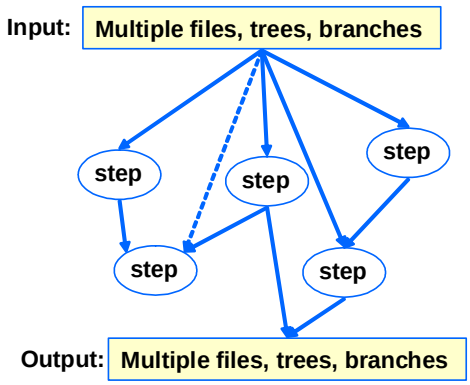

Figure 2: Analysis mesh (multiple IO).

The most flexible way would be to organize analysis steps like a mesh as shown in Fig. 2. Objects distributed in several files (branches) may be needed by several steps. Each step would specify the input objects it needs. The step itself should not care if these objects are already in memory (read or generated by some other steps), or if they have to be read by itself.

GO4 UPGRADE PHASE I

Many of the requiremnts mentioned above can already be met within the existing framework. Steps can be used in two ways: for object creation (provider steps) and for object processing (execution steps). An example setup is shown in Fig. 3. Only execution steps might produce output objects. These objects are never referenced directly as input by a subsequent execution step, but only by a (subsequent) provider step. Execution steps do never use their own input, but only the input of one or more provider steps.

With this strategy one can built dynamically configurable step meshes with multiple IO.

To be done

Currently the steps cannot share files. There must be one file per step IO. When the file management is moved to the framework steps could easily do IO on branch level. If a step hierarchy would be useful it can be easily introduced, just by deriving the steps from TTasks. The order of execution of the steps could be modified by an index table. This could be done also event by event.

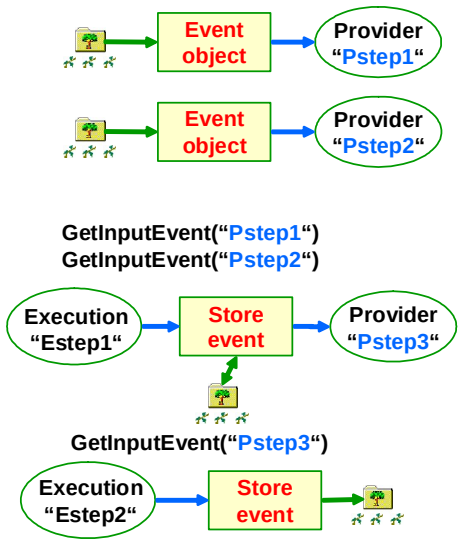

Figure 3: Provider and Execution steps.

Still Missing

Event stacking is not supported and cannot be provided easily, at least not with different depth on different input streams.

GO4 UPGRADE PHASE II

The approach of phase I gets inconvenient when too many steps are involved. The number of steps increases because of the provider steps. Therefore phase II introduces some new and more powerful design compoments.

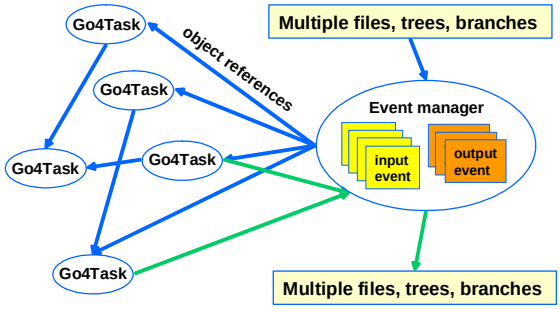

Figure 4: Event (IO) manager.

Go4 Event Manager

A new event manager (see Fig. 4) singleton is responsible for both the reading and writing of objects,

and for the control of the object exchange between steps (now Go4Tasks) in a mesh (smart pointer references). After setup the Go4EventManager can check the consistency of the object requirements and the execution order of the steps.

Go4 Task

A new Go4Task class combines the current Go4Steps with the ROOT TTask. Go4 tasks register to the Event manager. They subscribe for input objects. They register output objects. To interface the application layer to the framework a factory mechanism can be used.

The configuration of such an analysis mesh will be possible by macros or interactively by the GUI. New control GUI windows will be provided.

Autoconfiguration

Once the requirements of all tasks are known the event manager could create and enable automatically all tasks which are required by a specified one.

CONCLUSION

Because of the low efford the first phase can be set up soon, i.e. sharing files between steps and modify the execution order. A more general new design to set up meshes will be provided by phase II toghether with a better GUI control.

REFERENCES

[1] http://go4.gsi.de

[2] J.Adamczewski *et al*., "Go4 multitasking class library with ROOT", *IEEE Trans. Nucl. Sci*., vol. 49, pp. 521-524, Apr. 2001

[3] J.Adamczewski *et al*., "Go4 On-line Monitoring", *IEEE Trans. Nucl. Sci.*, vol.51, pp. 565-570, June 2004