

# Suez: Job Control and User Interface for CLEO III

Martin Lohner, Christopher D. Jones, Paul Avery

*University of Florida, Gainesville*

**Abstract.** Suez standardizes the way CLEO III data is processed by providing a common user interface for all CLEO III computing tasks. We present an overview of Suez design and implementation.

## INTRODUCTION

Suez, which stands for “Stream User interface made EaZy”, is the CLEO III job control and user interface for the CLEO III Data Access Framework, described in a separate submission to this conference [1].

Suez and the new Cleo III Data Access Model benefit from the history of the CLEO II data model, both in its short-comings and successes. One lesson learned from CLEO II and II.5 is that having to learn different user interfaces for different analysis frameworks (one for reconstruction, another for analysis, yet a third for calibration) severely hampers productivity. Underneath the different user interfaces the user finds different frameworks for accessing data. Each interface is specialized and well-tuned for its task and closely tied to the underlying data access API. Any commonality between the frameworks were not expressed in the interfaces.

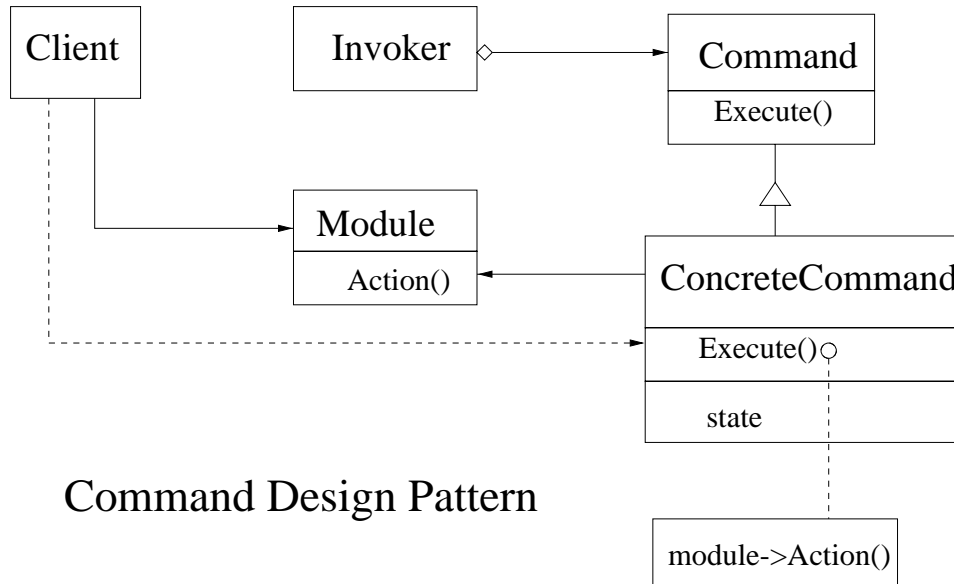
For CLEO III we wanted to learn from these lessons and provide one common CLEO III data access framework for reconstruction, analysis, calibration, and unify the user interfaces.

The maxims of the Cleo III Data Access Framework are reflected in the user interface design. All data are created equal and therefore to be treated equal; abandon any “event”-centricity. We have to support C++ and Fortran. The latter is of great importance, because we need to be able to reuse vast amounts of CLEO II reconstruction code. For fast development and flexibility we have to support static and dynamic linking and loading of “application” code. As another guiding principle we decided to insist on a clean separation between the interface and the data access framework. Furthermore, user feedback is of great importance; we rooted for early deployment and decided to support CLEO II analysis.

For an in-depth look at the CLEO III data access framework we refer you to a separate submission to this conference [1].

## BUILDING BLOCKS

The basic building blocks of Suez are *Modules* and the *Commands* which control them. The relationship between Module and Command follows the standard Command design pattern [2], shown in Fig. 1. Modules, as the name implies, allow for



**FIGURE 1.** Command and Module.

a very modular flexible system. Modules can be linked statically or dynamically. The Commands are registered with an *Interpreter*, described in more detail later.

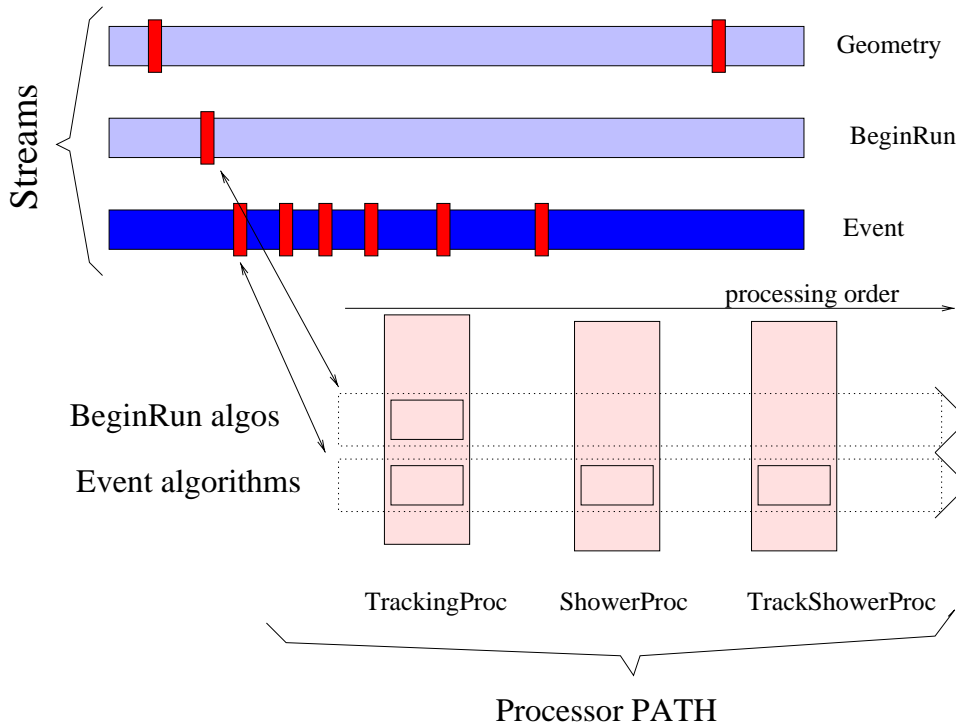
Physicists deal mostly with special types of Modules, called *Processors* and *Producers*, which are *containers of algorithms*. In the case of a Processor each algorithm is bound to a data *Stream* and executed whenever new information appears on that Stream. For instance, the "geometry constants" Stream would trigger any algorithms bound to that Stream in any selected Processor. The Producer module differs from the Processor module in that its algorithms are request-driven rather than Stream-driven. A Producer's algorithm is called automatically when the algorithm's result is requested. See [1] for further details.

A user can use Suez to select various Processors and Producers to assemble an analysis job. Either type of module can be linked statically or dynamically. Since the order in which Processors are selected matters, they are chained together in *Sequences* and *Paths*. Order does not matter for Producers.

Standard Commands and Modules in Suez handle loading of *Sourceformats* and *Sinkformats* (e.g. to be able to read a number of cleo-specific file formats and

the databases for event data and constants) and specifying which actual Sources and Streams to read from and Sinks to write to; loading and selecting Processors and Producers, control the “event” loop and setup histogramming. Processors and Producers may have their own Commands and Parameters to be able to setup and control a Processor’s or Producer’s behavior at runtime.

Since the order in which Processors are selected matters [1], Processors build an execution path, shown in Fig. 2. Currently we only support one path in the `suez`



**FIGURE 2.** A Path of Processors.

system, but further work is underway to allow for multiple paths, each one with their own output logic.

The Suez system is coded in C++. We encourage users to write new code in C++, but we have to also support a large amount of the existing Fortran-based reconstruction code which cannot easily be rewritten. A well-defined interface to Fortran allows us to do that.

## RUNTIME INTERFACE

The CLEO II user interfaces employ either home-grown command-line interfaces, or the CERN package “`kuip`”, or a number of very specialized graphical interfaces. For CLEO III we decided to use Tcl as the scripting language for our first prototype. We invited user feedback on other scripting languages (python, scheme, perl), but

Tcl was well accepted. Even though the current implementation uses Tcl, special care has been taken to hide Tcl code behind the Interpreter interface, in order to be able to bind to other scripting languages (e.g. python). Work is under way to build a GUI on top of the command-line interface based on Tk.

Since Tcl lacks GNU readline capabilities we had to implement a main Interpreter loop on top of the Tcl interpreter ourselves to allow us to use the GNU readline and history packages. This allowed us to support command-line editing, fancy history handling, and command completion.

## STATIC AND DYNAMIC LINKING

Suez supports both static and dynamic linking of Modules. The dynamic linking mechanism is based on the standard unix `dlopen` facility. To allow for platform differences we hide the details behind a `DynamicLoader` interface.

Standard loadable Modules consist of Processors, Producers, Source and Sink formats and various other specialized Modules (e.g. a module supplying a histogramming package). Suez itself is more or less a skeleton with pluggable parts.

No code changes are required to switch from a dynamically linked module to static linking, since we use factory methods [2] for creating the Modules. This means that even with static linking the module doesn't actually exist in memory, until it is selected, at which time its factory method will be called to create the module object.

For static linking the user needs to implement a central *UserApplication* class to register the factories for the Modules to be linked statically.

Dynamic linking only works without problems, if all libraries are compiled cleanly and kept properly up-to-date. We have encountered problems, whenever a user used old and new libraries together.

Dynamic linking provides for fast development time and quick turn-around on code changes (especially important for testing and debugging). For reconstruction we will most likely use all-statically linked executables to minimize the impact of inadvertent changes.

## SAMPLE JOB

To give a flavor for what an analysis job may look like, here is a sample job description:

```
unix> suez
Suez> source_format load DriverSourceFormat
Suez> sink_format load AsciiSinkFormat
Suez> producer select TrackProducer
Suez> producer interact TrackProducer
TrackProducer> parameter minimumHits 20
```

```
TrackProducer> exit
Suez> processor select AnalysisProcessor
Suez> processor select Cleo3DProc
Suez> database in ntrk>4 event
Suez> file in beginRunsFile.rp beginrun
Suez> file out eventSummaryStatistics.asc
Suez> go 10
Suez> exit
```

After loading certain file formats a Producer “TrackProducer” is loaded and setup to only consider tracks with more than 20 hits. The user’s analysis code is coded up as a Processor “AnalysisProcessor”, and depending on its setup, certain interesting events can be viewed with the graphics visualization Processor “Cleo3DProc”. The main event data are read in from the database with preselection to only read events with more than 4 tracks. Beginrun information is obtained from a file, and new data is written out to an ascii file (e.g. for debugging). As explained in more detail in [1], the TrackProducer will override the default tracks coming from the Source, in this case the database, and will for instance refit the tracks.

## CURRENT STATUS

We believe that user feedback is absolutely essential. We released Suez early in the development cycle and have received much feedback from both developers of CLEO III reconstruction software and by a number of physicists doing CLEO II data analysis.

For CLEO III reconstruction, we are able to read Monte Carlo events from file, load constants, reconstruct tracks and showers and other information, use a graphical display program [3] running as just another Processor, and write data out to a file. Reconstruction code consists both of new code, mostly written in C++, and legacy fortran code.

We support a reasonably diverse number of unix platforms, OSF1, Solaris, and Linux under both the native compilers on these platforms and the gnu compiler. Solaris is an important platform for various commercial software packages (e.g. Objectivity/DB).

We are close to employing a database interface (Objectivity/DB) both for event data and constants. Because of the abstraction of data Sources and data Sinks in the framework, the user can select at runtime which format(s) (a number of new and legacy file formats, or the database) to read data from and write to. No user code has to change to read/write different formats!

Suez will also be used in the online system with an OnlineSource as the input [4] to run fast reconstruction for data monitoring. The main reconstruction and data analysis will be run offline in the Nile [5] distributed processing environment.

## SUMMARY

Suez has grown from an early prototype with lots of user feedback to the standard user interface for the CLEO III Data Access Framework [1]. The goal to support one framework for reconstruction, analysis, and calibration has been successful, thanks to its modular structure and the flexibility of the underlying data access framework. The clean separation between Suez and the underlying data access framework has proven to be of great advantage.

This work is funded by the Department of Energy, grant DE-FG02-97ER41029.

## REFERENCES

1. M. Lohner, C. Jones, S. Patton, P. Avery, M. Athanas, *The CLEO III Data Access Framework*, CHEP98 Conference Proceedings.
2. Gamma *et.al.*, *Design Patterns*, Addison Wesley 1995.
3. C. Jones, P. Avery, *Spectator: A Reusable Framework for Information Displays*, CHEP98 Conference Proceedings.
4. C. Gwon, K. Honscheid, D. Hufnagel, J. Lorenc, H. Schwarthoff, A. Wolf, (Ohio State University), E. Lipeles, A. Shapiro, A. Weinstein, F. Wuerthwein, *The CLEO III Data Acquisition System*, CHEP98 Conference Proceedings.
5. M. Ogg, G. Obertelli, F. Handfield, and A. Ricciardi, *Nile: Large-Scale Distributed Processing and Job Control*, CHEP98 Proceedings.