CLEO's User Centric Data Access System

C. D. Jones[1]*, M. Lohner*[1]

Cornell University,USA

Abstract

When analyzing data, physicists want to spend their time studying the data instead of learning about and writing and debugging code for the data access system. We have tried to create a data access system that minimizes both the learning curve and code writing time.

The system is built on a simple data model we call the Frame. The Frame holds all the information necessary to describe the detector at an instance in time. All data is treated equally so a user can study changes in the accelerator's beam current the same way they would look for a decay in the collision events. In addition any piece of data can be retrieved from the Frame in a type-safe manner by calling the `extract` function. This design means once a user learns how to access one type of data, they can access any data.

The program Suez is used to provide a Frame to the user's analysis code. Suez is a lightweight program which can dynamically load modules that can be used to read/write data, create new data, or analyze data. The flexibility and efficiency of the Suez system allows us to use one program for all of our data access needs: software trigger, online monitoring, calibration, reconstruction, Monte Carlo generation, analysis and event display.

Keywords: data access, user friendly, usability, Frame

1 Introduction

When analyzing data, physicists want to spend their time studying the data instead of learning about, writing and debugging code for the data access system. We have tried to create a data access system that minimizes both the learning curve and code writing time. We accomplished this by creating a simple data model, a general purpose data access framework, easy to write and auto-generated code, and an easy to use interface program.

2 Data Model

The first step we took in designing the data access system was to develop the mental model we wanted the physicists to use when working with this system. We wanted the mental model to be simple, general enough to encapsulate all data associated with the experiment (e.g. both event and calibration data) and based on ideas physicists already understand. These requirements were satisfied by the Frame-Stream model [1].

The essential elements of the Frame-Stream model are shown in Figure 1. The Frame-Stream model presents data in a manner analogous to how data is collected in the online data taking system. The online system provides streams of time-ordered data records. The Frame-Stream model has the concept of a Record which holds data that are related by life-time. So the Event Record holds all information that is relevant for the instant the event occurred, e.g. the raw hits in the detector and the tracks reconstructed from those hits. A Stream is a time ordered set

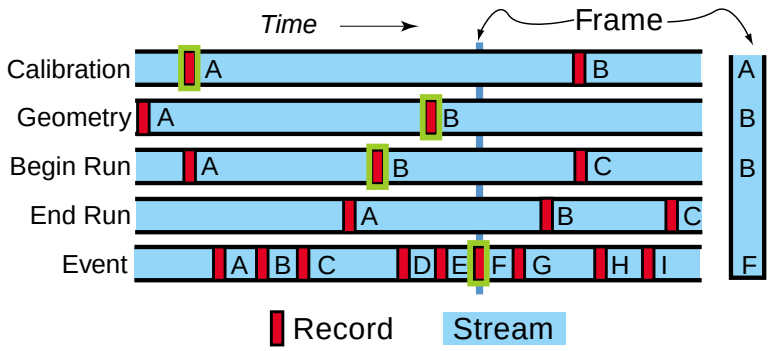

Figure 1: The Frame (shown on the right) is a collection of Records that describe the state of the CLEO detector at an instant in time.

of Records, e.g. the Event Stream holds a time ordered collection of Event Records. The new concept in this model is the Frame, which is a snapshot of the CLEO experiment at an instant in time which is formed by the most recent Record in each Stream.

To do analysis in the Frame-Stream model, a physicist specifies algorithms he wants to have run when new Records appear in particular Streams. For example, if a physicist wants to to do an Event analysis, he would specify that his algorithm should be run whenever a new Event Record appears in the Event Stream. Similarly, if a physicist wants to do an analysis of the accelerator's beam currents she would specify that her algorithm should be run whenever a new Accelerator Status Record appears in the Accelerator Status Stream. This generality means that once a physicist learns how to do one type of analysis (e.g. Event) they can directly use that knowledge to do any other type of analysis.

3 System Concepts

The Frame-Stream model explains how physicists should think about the data and outlines how analysis is done. The model does not say how a data analysis job is setup nor does it say how data are actually created and transferred to the physicist's algorithm. These items are defined by the data access system. The goal of the data access system is to be able to handle any job that requires access to the data so physicists only have to learn one system. The list of jobs includes: physics analysis, event display, reconstruction, Monte Carlo generation, calibration, online data monitoring and online software trigger. This goal was realized by making the data access system a light-weight framework which uses modular components to do the work. Therefore physicists construct their jobs by selecting the appropriate components to run in the data access framework.

A schematic overview of the concepts used in the data access system is shown in Figure 2. There are two broad categories of components in the system: Data Providers and Data Consumers. Data Providers put data into the Frame and Data Consumers read data from the Frame, therefore the Frame works like a data bus. The data bus analogy is very apt because Data Providers do not directly put their data into the Frame. Instead, when a Data Consumer asks for some data from the Frame, the Frame finds the appropriate Data Provider and then gets the data from the Provider. In this way the system only does the work necessary to access data when that data is requested.

There are two types of Providers: Sources and Producers. Sources are the origins of Streams, that is only a Source can say when a new Record is supposed to appear. Most Sources

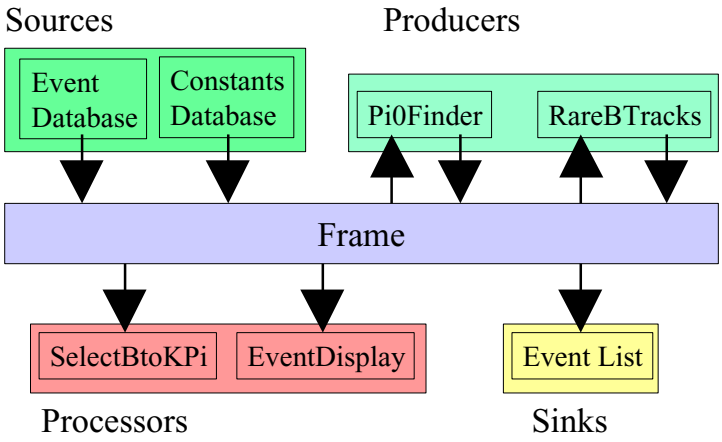

Figure 2: In the data access system the Frame works like a system bus. All communication between components is handled via the Frame.

read their Records from a persistent store such as a file or a database, but we also have a Source that gets Records directly from the online data taking system. A Producer is an encapsulation of an algorithm (or related algorithms) that produces new data. For example, π^0's are found by a Pi0Finder Producer. Producers usually need additional data to do their work; in the previous example the Pi0Finder needs access to hits in the Calorimeter. A Producer accesses this information using the Frame in the same way that Data Consumers access the information.

Data Consumers want to process all Records from particular Streams. There are two types of Consumers: Sinks and Processors. Sinks access Records from the Frame and then write the data in those Records out to a persistent store. Usually Sinks write out what Sources read in. A Processor is a user algorithm that wants to run a calculation on all Records in particular Streams. E.g. An Event analysis is a Processor that wants to get data from all Event Records.

A physicist's data access job is composed of a series of Sources which supply the Streams of interest, Producers which create additional data that is not stored in the Sources, Processors which perform the physicist's algorithms and finally Sinks which store the data for later use.

4 Writing Code

Physicists want to spend their time writing code to implement algorithms and get frustrated if they have to spend large amounts of time writing and debugging code to access data.

To reduce the burden of using the data access framework, we provide skeleton code generation programs. For example, if a physicist wants to write a new Processor named MyProc, all they have to do is type "`mkproc MyProc`" and the proper directory structure, Makefile and skeleton source code files are generated. The physicist merely has to write her algorithm in the member function skeleton that is provided. The only data access framework code the physicist might write is one line if she wants to process a Record other than the Event Record, because the Event Record code is automatically written by default. To do this a physicist adds to her Processor's constructor code that sets which member function to call when a new Record appears in a certain Stream, e.g.

```
bind(Stream::kGeometry, &MyProc::geometryChanged );
```

We wanted to make sure that it is impossible for physicists to access non-existent or stale data. To that end, physicists can only access data by calling a function that reads the data from a

Record. If the data is not available, a C++ exception is thrown. We have found that throwing an exception is preferred to returning a null pointer because physicists often do not check the validity of the returned pointer. In contrast, only if a physicist's code can deal with missing data (which is rare) do they need to write code to catch an exception. Any non caught exception will be caught by the data access system and a helpful error message will be printed before exiting gracefully. Therefore diagnosing a problem is much easier with exceptions than with null pointers.

To access data one must do the following

```
MyProc::event( Frame& iFrame ) {
 ...
 FAItem< EventHeader > eventHeader;
 extract( iFrame.record( Stream::kEvent ), eventHeader );
```

`FAItem<>` is a smart pointer to a singly occurring item where the template type is the type of data the physicist wants to obtain. We use `FAItem<>` so that no one is tempted to delete data they have extracted. `extract` is a templated function which extracts the requested data from a Record in a type-safe manner. Using a templated function means one only has to specify the type of data to extract when one defines the variable to hold the data.

To extract a table of data, say a list of Tracks, one has to write:

```
 FATable< Track > tracks;
 extract( iFrame.record( Stream::kEvent ), tracks );
```

`FATable<>` is a smart pointer for our own container class `FAPtrTable<>`. We have our own container class to enforce our policy that every multiply occurring object must have a unique identifier. E.g. each Track in a `FAPtrTable<Track>` has its own unique number. This allows a physicist to say "track 4 is a muon" and everyone will agree on which track is track 4 even if they have taken some tracks out of the master list and placed them in a different container.

5 Running a Job

The user interface to the data access system is provided by the program Suez [2]. Suez provides a command line interface with a Tcl interpreter and provides tab completion and command history similar to most Unix shells. Suez allows either static or dynamic linking of the components. Dynamic linking has been extremely successful, it has allowed users to reduce linking time from potentially minutes to just seconds.

6 Conclusion

From the feedback we have received from our users, it appears that the system is easy to learn to use. After a short initial introduction to the system, users tell us that it is very intuitive.

The biggest usability problems remaining are: users do not know what data types are available to extract from the Frame and users do not know which components need to be loaded to make their jobs run. The former problem we are correcting by providing better documentation. The latter problem we are correcting by determining exactly what types of data each component produces and consumes and then using that information to help physicists assemble their jobs.

References

1 P. Avery, C.D. Jones, M. Lohner, S. Patton, "Design and Implementation of the CLEO III Data Analysis Model", CHEP'97, Berlin, Spring 1997.

2 M. Lohner, C.D. Jones, P. Avery, "SUEZ: Job Control and User Interface for CLEO III", CHEP'98, Chicago, Autumn 1998.