CLEO’s User Centric Data Access System

Christopher D. Jones
Cornell University
Introduction

Physicists want to spend their time studying the data instead of learning about, writing and debugging code for the data access system. Our goal was to create a data access system that minimizes

- learning curve
- code writing time
- code debugging time

Guiding Principles

- concentrate on how physicists use and think about data
- use general purpose ideas so users can learn one thing and apply it everywhere
- do not lock a physicist into one data storage format
- if it is hard to use, it is our fault and we should fix it immediately
A **Record** holds all data that are related by life-time

  e.g. Event Record holds Raw Data, Tracks, Calorimeter Showers, etc.

A **Stream** is a time ordered sequence of Records

A **Frame** is a collection of Records that describe the state of the detector at an instant in time.

A user’s analysis states what Streams she wishes to study (e.g. Event) and at each time a new Record appears in one of those Streams we give her a Frame.
Processing Data

- A new Record is read in from a Source
- A new Frame is created holding all the Records pertinent to that instant in time
- The Frame is passed to Processors which decide if the Record is interesting
- If the Record is interesting to all Processors, it is written out to a Sink
General Purpose Access Framework

Data Providers: data returned when requested

Sources: data from storage

Producers: data from algorithm

Processors: analyze and filter data

Sinks: store data

Data Requestor: sequentially run requestors for each new Record from a source
Multiple Sources and Sinks

- Can read from/write to multiple Sources/Sinks at same time
- The multiple Sources and Sinks can be different formats
  - Read user data from a file and reconstruction from database
- Only one Source can be used to decide exactly what Records (e.g. Events) to process
  - Sources that determine what to process are called Active.
  - In above example, system will prompt you to determine if you want to run over all the events in the user’s file OR in the database.
- CLEO uses many different formats
  - Several data file formats
  - Objectivity database for event storage
  - CORBA data transfer from DAQ
  - Calibration Objectivity database
  - Calibration text file formats
  - Etc.
Easy to Write Correct Code

Try to reduce the time it takes to write and debug code

**Skeleton Code Generators**
- use scripts to generate source code and directories
  ```sh
  mkproc MyAnalysisProc
  ```
- user just writes her algorithm in the member function skeletons

**Cannot change data once it has been published**
- avoids accidentally corrupting data that may be used by other processes

**Cannot get ‘bad’ data**
- data can only be accessed by calling a function that retrieves data from a Record
  - I.e. no global data
- an exception is thrown if no data is available

Why we use exceptions instead of returning a null pointer

- **null pointer**: even if a user must have the data for his algorithm to work, he must check for a null pointer and ‘do the right thing’ if it is null. If they do not check, the program will crash and he will lose all of his work

- **exceptions**: only if a user can deal with missing data (which is rare) do they need to catch the exception. Any uncaught exception will be caught by the data access system and a useful error message will be given before the program exits
Simple Data Access Call

All data is accessed through a type-safe call to the templated `extract` function

```cpp
MyAnalysisProcessor::event( Frame& iFrame ) {
    
    Item< RunHeader > runHeader;
    extract( iFrame.record( Stream::kBeginRun ), runHeader );
    
    Table< Track > tracks;
    extract( iFrame.record( Stream::kEvent ), tracks );
```

**Item<>** is a smart pointer to a singly occurring item, where the template type is the type of data the user wants.

**Table<>** is a smart pointer to our own container class. We use our own container class to guarantee that all multiply occurring items each have their own unique identifier.

If say ‘track 4 is a muon’ everyone agrees on which track is # 4 even if it appears in multiple lists.
One data access executable for all our data access needs

Physics Analysis, Event Display, Reconstruction, Monte Carlo Production,
Online Data Quality Monitoring, Online Software Trigger

Users only need to learn one program

Modular

the executable is a light-weight framework which allows code ‘modules’ to be
assembled to form a particular data access job

Dynamic Linking

drastically reduces link time

easy comparison of two algorithms that produce the same data

statically linked executables are also supported
Command Line Interface

• Fully scriptable via Tcl interpreter
  – e.g. can create variables and make loops
• Full ‘shell’ like editing
  – Tab completion of command name or file name
  – Use left and right arrows to move cursor on the line
  – Use standard shell command keys
    • e.g. <control> k: delete to end of line
• Full history
  – Use up and down arrows to scroll through command history
  – history : returns list of all commands ever typed
    – etc.
• All commands support ‘help’ sub-command