Developing CLEO III Online Data Analysis Software

Heather Smith

Department of Physics

University of Illinois, Urbana-Champaign, Illinois, 61801

Abstract

The current CLEO III hardware and software upgrade includes the replacement of KANDI, the Kansas Display, that was previously used to display plots of the data collected at the end of each run of the detector. KANDI depended on accessing data through the CERN libraries and using a specialized graphics library, HIGZ. The data from the detector for CLEO III will be stored in a database, instead of the CERN libraries, so the decision was made to completely rewrite KANDI in two parts using CORBA. The first part is written in C++ and is required to retrieve data from the database and format it so that the data can be plotted by software written in Java. The C++ code is written, but has yet to be tested in conjunction with the Java code to observe how well the two pieces of code communicate with each other to accomplish the desired objective.

Introduction

One of the most important indications of what the detector is doing and what kind of data are being produced is the plots displayed by the online data analysis software. It is extremely useful to obtain a running status report of what data is being produced to make sure the detector is working properly. If periodic checks are not made, then several runs of garbage could be collected that would simply be a waste of time and energy—hence the motivation to develop and maintain useful, reliable online data analysis software.

Figures 1 and 2 show examples of KANDI plots. The histograms produced by KANDI usually display the variable that is being examined, such as beam energy, versus run number so that one of the people monitoring the detector can see how the variable changes over time. There are usually specified limits that the variable must remain within or there is a serious problem. If one of the people monitoring the detector notices that a key plot is significantly different than expected, an expert is called who will know how to deal with the situation.

KANDI was the online data analysis software developed for CLEO II. Now CLEO is undergoing significant hardware and software upgrades that require at least a modified version of this existing code. At least part of the motivation for updating the KANDI code is that it was written in Fortran, and there has been a concerted effort to convert most CLEO III code to C or C++.

However, one of the more important reasons for essentially replacing KANDI is that KANDI is designed to retrieve data to be plotted from the CERN libraries, and data for CLEO III will be stored in a database instead of in CERN library format. So, all of the calls to obtain data to plot would have to be completely rewritten to fit the new database format.

Also, KANDI made use of HIGZ, a specialized non-standard graphics library to plot the data. Anyone revising the KANDI code would first have to familiarize him/herself with the HIGZ library—which would never be applicable to any other project. It was decided to

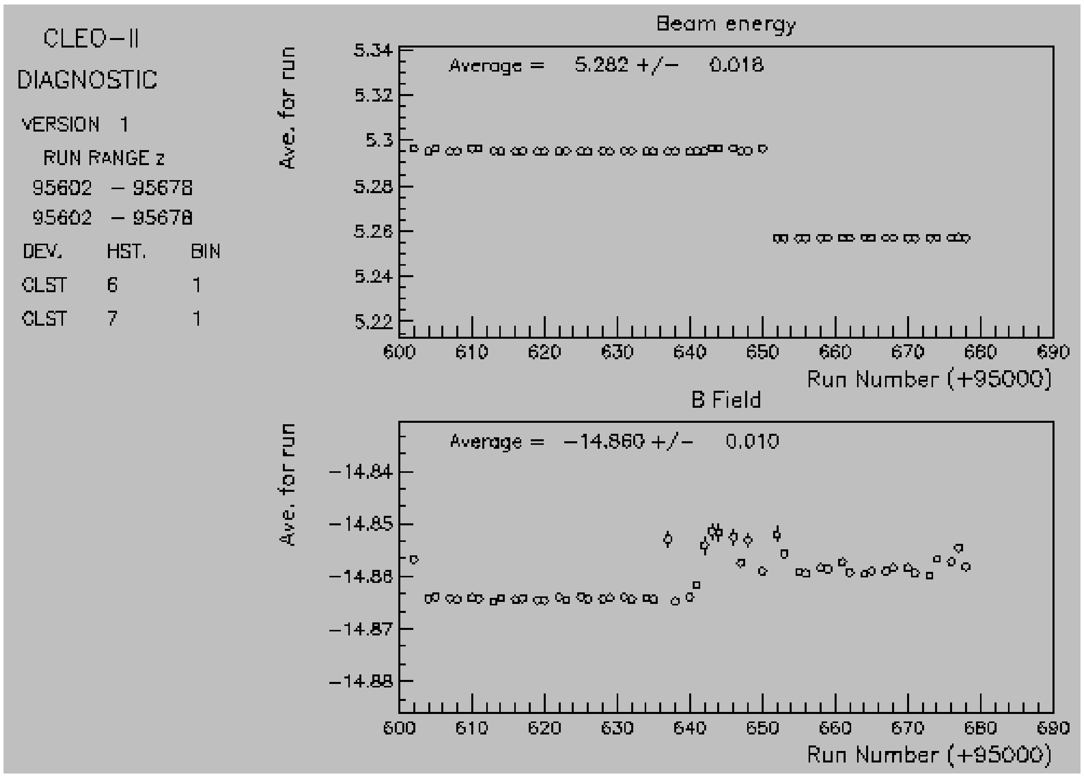

FIGURE 1. (top) The beam energy is plotted as a function of run number. The on-resonance beam energy should be 5.2945 GeV, give or take 0.5 MeV. If the beam energy drifts outside of this range please ask the CESR operator to make the appropriate energy change. (bottom) The B field is plotted as a function of run number in units of 1000 Gauss. 10 Gauss fluctuations (0.01 units) are normal. [1]

simply scrap KANDI—on the basis that it would take more time to try to salvage it than it was worth—and start fresh using C++, CORBA, and Java to write a whole new software package for displaying data while the detector was running. This even makes it simpler to add new features to the software package.

Improving on KANDI

One of the most salient improvements of the new software design over KANDI is that—because of CORBA's built-in flexibility—the plots of the data may be viewed from any computer linked into the system. This means that experts who are on call in case of emergencies can, when rousted out of bed at 4 AM, look up the questionable plot on their home PC instead of being forced to march all the way in to the Counting Room where KANDI was permanently stationed. This makes the new software more convenient and generally flexible than the previous KANDI code had been.

One of the other features discussed was allowing the user to create plots that were completely user-specified, instead of being limited to a predefined set of plots. This feature could be useful if there were a specific plot that might be important to look at only once in a very

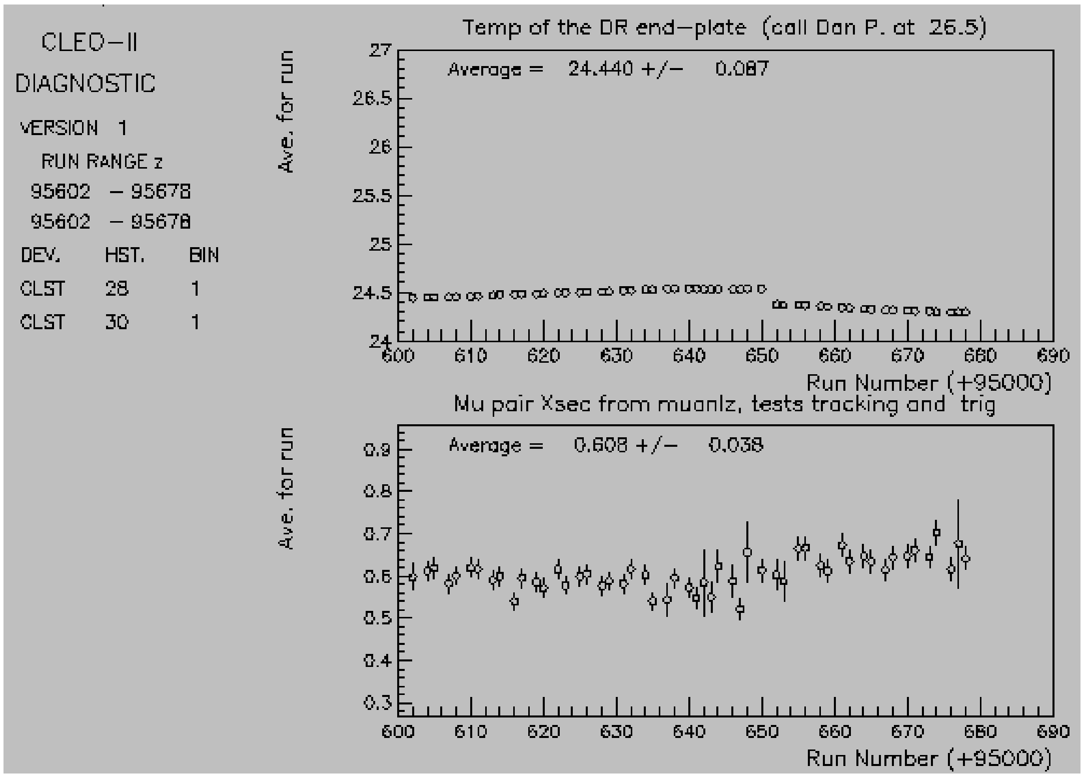

FIGURE 2. (top) The temperature of the DR endplate is important because wires may break if it gets too hot. (bottom) A change in the mu pair cross section might indicate a problem with the trigger or tracking. Look for shifts in other Kandi plots and then call the appropriate expert. [1]

great while. Providing for this possibility gives the code more flexibility to adapt itself to unusual circumstances.

Writing the code in standard C++ and Java makes the software easier to debug and maintain over time. It is more readable and universally understandable so that it can be adapted fairly quickly by anyone with sufficient background in most well-known programming languages. One of the most significant worries about KANDI was that it was hard to maintain since very few people could—or wanted to—decipher the HIGZ graphics library calls and learn enough about the code to keep it working.

The new software even allows plots to be generated from calibration runs, which will be useful as all of the upgrades are installed and tested. KANDI, on the other hand, would only plot data runs.

These are some of the more obvious advantages the new software has to offer in its task of replacing KANDI.

CORBA

CORBA is, essentially, simply a bridge between two pieces of code, a server and a client. The server and the client do not necessarily have to be on the same machine—hence its portability and flexibility. CORBA allows the server to communicate with the client through an *idl* (interface definition language). The *idl* specifies which functions and parameters both pieces of code can access. The server and client do not even have to be written in the same programming language—in fact, they often are not. CORBA allows the programmer to write each part of the software in the language that handles it best. For example, Java is a very useful language for writing GUI's and displaying graphics—like plots of data. However, Java can be somewhat slow and cumbersome in areas where C++ excels. By giving the programmer the ability to write code in the language that performs best in that particular area, CORBA allows one to write the most efficient code possible to perform any given task.

CORBA also has very useful features like allowing the programmer to declare a variable of type `Any` in the *idl*. That way, when the *idl* is being written, if the type of the specific variable could be one of six or more possibilities, the programmer does not have to write a large `case` statement or overload a function for the several possible types of the variable. CORBA also allows the programmer to define `sequence`s that are variable in length. CORBA makes a lot of the details of a complicated project simpler by handling some the grunt-work on its own.

For this software package, Java is being used to actually perform the graphics manipulations and plot the data and C++ is being used to access the data and send it on to the Java program. In the *idl*, two interfaces are defined, the `RunHist` and `UserHist` interfaces. The `RunHist` interface deals with standard set of plots that must be displayed at the end of every run. The `UserHist` interface deals with specialized plots that may useful if the standard plots are not displaying the data required for the particular situation.

Interfaces

The interface that will be used most frequently is the `RunHist`. There are three functions in the `RunHist` implementation (written in C++). The first function gets a list of histograms to create from a script and assigns them a queue number to keep track of which request should be processed when. The final format of the aforementioned script is yet to be determined. However, its general purpose is to provide a list of specifications for creating Histogram objects in the form of a `SpecTableEntryList`. The next function is called with a set of specifications for creating a histogram object to plot. It fills the histogram with the requested data and returns a histogram object with the appropriate number of axes for the number of variables and other such minor, but important, details. The last function sends a maximum of four histograms at a time to be plotted on a page. It looks for the next list of histograms on its list to be processed and keeps sending them off four at a time (or less, if there are fewer than four left to be sent) until there are none left to send.

Filling a histogram object with the requested data involves a lot of handshaking with the database itself, so there are two separate classes that deal with extracting the correct data from the database. The first class is the `DataCache` class that is mostly useful for `RunHist` functions and implementations. This class keeps a cache for each of the variables in the `RunHist` script so that the data does not have to be transferred out of the database every time a plot is popped up in a window. This way, the values of the variables that will be used over and over again are stored for the last X number of runs in a specific place so that less time will have to be spent extracting data. Each time a new run is finished, the cache for each variable is updated and new plots are made. This class can also be useful for `UserHist` plots because the variable to be plotted may already be, at least in part, cached. For example, in the cache, only the values for the last fixed set of runs are kept around. If a shifter wanted to (ok, had to...) look up some runs that are not in the cache and compare them to the ones in the cache, then half the values are already found and waiting to be plotted. The `DataCache` class, in general, is a useful time and CPU saver.

The second class that handles getting information out of the database itself is the `DataRetrieval` class. It creates and deletes data caches. It fetches all the data for a particular run number and then fills each data cache with the values for its variable for a specified run range. It will also find the data for user defined histograms. If there is no data cache for the variable, then it finds the data anyway and sends it back to the `UserHist` interface.

The `DataCache` and `DataRetrieval` classes do not actually touch the database themselves because there is yet another layer of masking in between. The two classes developed actually contact the `retrieve()` function in the *DBRunStatistics idl* to make the direct calls to the database. Since the `retrieve()` function returns all the data from the database for a particular run number, the `DataCache` and `DataRetrieval` classes only call that function once every time a plot needs to be updated. When the data caches need to be filled, initially, then the `retrieve()` function is called several times (once for each run number) and the `RunSummary` functions loop over the data that needs to be initialized. That way, all the data for one run number is retrieved, then all the variables that needed to be updated for that run number are stored in the data cache, and the next influx of data is retrieved for the next run number and so on until all the data caches are filled.

The `UserHist` interface gets one set of specifications for a histogram and does a custom version of what the `RunHist` interface does. The main differences between the `RunHist` and `UserHist` interfaces are that the `RunHist` interface keeps data cached so that it will save time and the `UserHist` interface produces one plot at a time (not many from a script) and does not need to be as efficient as the `RunHist` interface. The `RunHist` interface will be run much more often, so it needs to be faster and more efficient.

Conclusions

The C++ portion of the Online Data Analysis software that will replace KANDI for CLEO III has been written. Most of it has been tested to the extent that it can be without coordinating testing with the Java portion of the software. The C++ code provides functionality and flexibility and attempts to do its task quickly and efficiently. A lot of thought has been put into the functions and classes written to handle the data and process it for the Java program. The *RunSummary idl* has been written and rewritten in an attempt to make sure that it provides enough functionality for the Java code so that it will be able to successfully plot the data passed to it from the database. The `RunSummary` program uses several *idl*'s for various purposes to orchestrate the smooth flow of the code. The *Histogram idl* provides a set of parameters to be filled that describe what is required to plot a histogram. The *DBRunStatistics idl* actually makes the calls to the database and returns all of the data for one run number. The *RunSummary idl* dictates how the C++ code will communicate with the Java code to actually produce plots.

What remains to be done is to actually test the C++ and Java code together to be sure that they coordinate well with each other. Also, a great wealth of comments will be added so that the code not only works reasonably well, but is also maintainable and readable for those who need to modify the code in the future.

Acknowledgments

I would like to acknowledge Dr. Andreas Wolf from Cornell University for proposing this research project and his endless patience as I ploughed through new realms of C++ and CORBA. I would also like to thank all of the graduate students, postdocs, and professors in the Illinois office who were always willing to answer my questions and provide guidance.

Of course, I cannot forget to thank my fellow colleagues in the REU program who inspired me with their diligence and perseverance, and all of the faculty and staff who made this REU program at Cornell University so very enjoyable, entertaining, and instructive.

This work was supported by the National Science Foundation REU grant PHY-9731882 and research grant PHY-9809799.

Footnotes and References

1. CLEO Run Management, "GLOBAL Histograms", http://lnson3.lns.cornell.edu/doc/kandi/node3x.html