Software Development for Linear Accelerator Data Acquisition Systems

Jackson DeBuhr
Department of Physics, Applied Physics, and Astronomy,
Rensselaer Polytechnic Institute, Troy, NY, 12180
(Dated: August 12, 2004)

The TESLA Test Facility at DESY in Hamburg is a research and development facility demonstrating superconducting technology for a linear accelerator and a prototype free electron laser in the vacuum ultraviolet range. An upgrade to the facility was recently completed and a data acquisition system (DAQ) is undergoing development. Key components of the data output end of the DAQ were completed including routines to handle the concurrent output of the data into different formats that will ultimately be the primary means through which the machine data will be analyzed. At the conclusion of the project every component vital to the DAQ data flow has a solution and the data can make it from the front-end devices to tape storage for later analysis.

I. INTRODUCTION

A linear collider is considered to be the next project that the high energy physics community needs. There are many aspects of high energy physics that could be explored with a linear collider, including such things as supersymmetry and the Higg's Mechanism. Additionally, the links between elementary particle physics and cosmology are well recognized and the discoveries made about the small scale world of elementary particles could tell much about the entire universe. The technical challenges to such a project are large. For instance, some of these difficulties include accelerating the beams to high energy and attaining desirable beam characteristics.

Currently two methods for the beam acceleration are being considered, the so-called "warm" and "cold" options. The "warm" option would use conventional accelerator cavities while the "cold" would involve superconducting cavities. Again, there are many technical challenges to creating superconducting accelerator modules, one example of which is the low temperature at which they operate: about 4 K. The Laboratory for Elementary Particle Physics (LEPP) at Cornell University is a member of the TESLA collaboration, which is a group working on an implementation of a future linear collider facility. In addition to the work done with superconducting acceleration cavities, LEPP is also involved with developing a data acquisition system for the TESLA Test Facility (TTF2), where various modern technologies for control and acquisition systems are deployed and put to use.

TTF2

TTF2 is an accelerator prototype at the DESY site in Hamburg Germany which demonstrates, among other things, superconducting technologies for RF cavities, and a free electron laser in the vacuum ultraviolet range (VUV-FEL) based on the principle of self amplified stimulated emission (SASE)[1]. Though the TTF2 project has turned their emphasis more

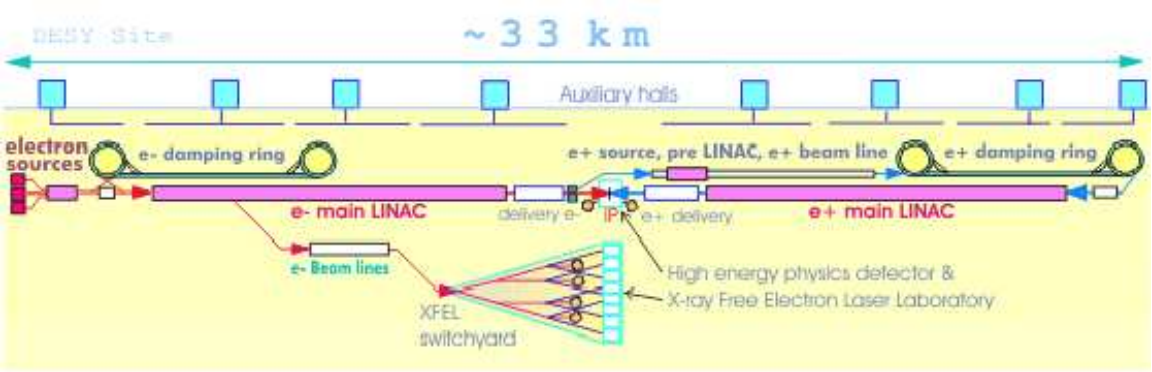

FIG. 1: Layout for the proposed linear collider

toward the VUV-FEL, the research done with the superconducting acceleration cavities and the beam characteristics will prove very valuable for a future linear collider project. The 250 meter long beam line carries scores of diagnostic and control devices that will be used to study such properties as the emittance and size of the beam, both of which would be very important factors to optimize in a linear collider.

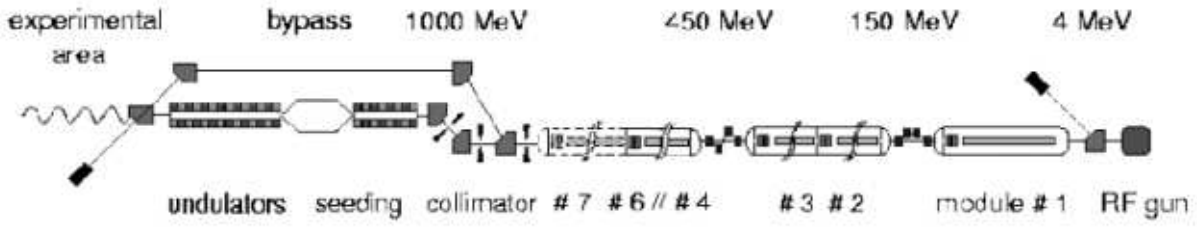

FIG. 2: Layout of TTF2

Another aspect of TTF2 project that deserves mentioning is that it is also prototyping the Global Accelerator Network (GAN) concept[2]. The scale and complexity of future plans in the high energy physics community has grown beyond the point where one institution or even one country can hope to complete all of the necessary work in any reasonable timeframe. Recognizing this, the GAN concept was formed so that the best use could be made of the ideas, resources, and centers of excellence in accelerator physics for completion of a linear collider project.

The TTF2 DAQ

The problem that the TTF2 data acquisition system (DAQ) must solve is how to get data from all of the various components into some useful form. The system that has been devised is based highly on the DOOCS[3] framework. DOOCS, which is a Distributed Object Oriented Control System, allows the data acquisition to work transparently between computers across the local network or even from off site. This remote capability built into DOOCS follows well with the GAN concept. All of the components along the DAQ flow are implemented as DOOCS servers. This allows a high degree of modularity as the interface

between DOOCS servers is very well defined. Another important overall feature of the DAQ system is that an object oriented approach was utilized to increase the modularity and ease of maintenance of the entire system. With object oriented code, a small piece of the system can be rewritten without effecting the rest. Portability is another possibility of object oriented code. What this means is that an existing system can be used at different sites or experiments with very little work.

To gain some understanding of the design of the DAQ some data will be followed from its conception in the devices until it is written to disk for later analysis. As an example of one data source at the TTF2 site, consider a stripline beam position monitor (BPM). At the TTF2 site an RF gun is the source of the electrons in the beam. A laser is turned on briefly, and during this time, electrons are ejected from the photocathode of the gun. These bunches are created many times in rapid succession forming what is know as a macropulse. As the bunch passes the BPM, a signal is produced in the electronics of the device. A dedicated computer, which is running a DOOCS server, is located near all of the devices along the beamline. The purpose of this server is to convert the signals the BPM detects into a digital form. These ADC servers are the first step in the DAQ. In addition to the digitizing of the data coming from the devices, DOOCS affords the ability to run filtering algorithms, fitting routines and other processing capabilities that can be performed on the raw data. The raw data along with any of the processed information is sent by the server to the next major component of the DAQ: the Shared Buffer Manager. This Buffer Manager acts as a repository for the numerous servers that provide data from the machine. While in the Buffer Manager, the various data is sorted and reconstructed into an event record. Essentially, these event records contain all of the data the machine outputs for a given macropulse. These event records are then sent over the network to the Event Builder, the last major DAQ component. The Event Builder is responsible for taking the event records and converting the data into a more usable format for analysis. Additionally, the Event Builder is capable of processing and filtering events and therefore behaves as an additional software-based filter level. This data is then written to disk using the storage system dCache[4]. This system provides efficient storage and retrieval of the large quantities of data produced in accelerator facilities. One important feature of dCache is that it is GRID[5] enabled, which is becoming increasingly important for handling and analysis of massive data sets.

II. EVENT BUILDER

The work of this project focused on completing components of the Event Builder. The Event Builder is the critical step in the DAQ process, because without it the data would not be very accessible to the general user. Though the event records are usable, the author can assure that it is not a trivial job to work with this format. So to facilitate the analysis of the data, the Event Builder converts the event record into a usable data format (see Fig. 3). Like every component in the DAQ the event builder makes heavy use of DOOCS and object oriented design techniques. The object oriented approach is not only cleaner conceptually, but it affords easy maintenance, upgrade capability and portability.

At the beginning of this project the Event Builder was still missing some major pieces of its functionality. The link with the Buffer Manager had not been completed. Also, the data output routines had not been started. Though much work had already been put into the Event Builder, these two ends of the data flow had not been implemented. The goal of this project was to complete the link with the Buffer Manager, and to provide an implementation

for the data output. With these components in place, the Event Builder could be run, and the entire data flow of the DAQ would be completed. This would mean that data would be able to get from the devices all the way to the tape archive for later analysis.

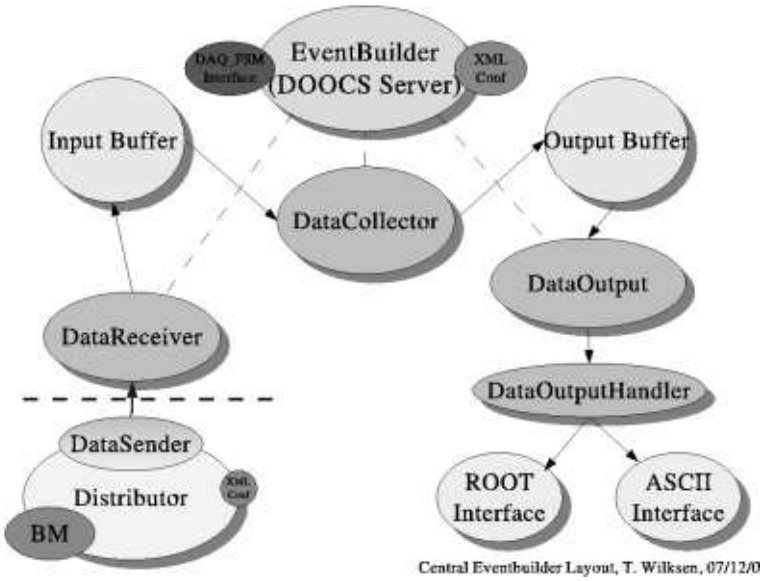

FIG. 3: An overview of the design of the Event Builder

III. DATA LINK

The first piece of the Event Builder that needed work was the link to the Buffer Manager. Without this link the Event Builder would not receive any data and could therefore not provide any for the users. This component was a relatively straightforward application of the UNIX socket implementation. In keeping with the object oriented design of the DAQ, two objects were created to realize this link. One of them, the sender, was responsible for contacting the receiver and sending the information. The other object, the receiver, connects to the sender and waits for the information.

IV. EVENT PARSER

The event record format used in the Buffer Manager is not conducive to easy analysis. An object was created that took data in the Buffer Manager format and parsed the data. In other words, this event parser, takes the event record and provides a kind of handle to each of the logical components of the record. This was a very important step for all of the Data Output routines described below. Additionally, having one object responsible for parsing the data will allow an almost trivial update if the event record format is ever changed.

Functionality was added to the Event Parser to print the event records in an easily readable form. This allowed some debugging work to take place including catching some errors in the event records themselves which the Buffer Manager's maintainer promptly fixed.

V. ROOT DATA OUTPUT FORMAT

The main purpose of collecting machine data is so the experimenters and operators can analyze the recorded data. There are many analysis packages that are widely used, and it is possible that eventually output routines will be written for each of these packages, but for the beginning it was decided to implement a data format for the analysis package called ROOT[6]. One of the major concepts that ROOT provides is the "Tree object." A ROOT tree is essentially an ntuple, with the columns of the ntuple given the name branches. One feature of trees are their flexibility which allows a much more powerful and dynamic data representation.

There was much discussion about the format that will be used for the ROOT output. In fact, the ROOT format was always a major point during the collaboration meetings the author attended during the project. Care needed to be taken as it will be used by everyone. The design that was decided on was a multiple tree organization. There are first two general classes of data: machine data, which is anything that comes from the machine controls or diagnostics, and the user experiment data. Each of these types then will have two subtypes, the so-called fast data and slow data. The fast data is anything that has a repetition rate equal to the rate of macropulses. For instance the signals detected in the BPM's are fast data. Slow data, on the other hand, generally has a slower update time. One example of slow data is the current in a given magnet. It was decided that there will be one tree for each of the types: a fast machine data tree, a slow machine data tree, a fast user experiment tree, and a slow user experiment tree.

Each of the trees will share a similar layout. There will be one branch that holds the event header. This is information such as a run number, the time of the event and the macropulse number. Then, there will be some number of branches that hold the server block header information. Information contained in the server block headers includes, for example, which channels a server provides and the trigger mask. This data is less useful to the general user, but may be important for technical debugging of the data path. Then there will be some number of branches holding the actual channel data. An additional ROOT mechanism will be used to connect the various trees together so that analysis may use data from all of the trees.

Given this scheme, the actual objects that will be contained in the tree needed to be defined and implemented. A first version of the object definitions were created that were focused on the channels as the basic object type. So a regular beam position monitor, which has two channels representing the x and y positions of the beam, would have two objects in the tree and two branches. After some discussion with the group and some helpful hints, a new version was made which was centered on devices, but which also made use of another ROOT feature called splitting. This splitting can make one branch act also as if it were many sub-branches. This scheme allowed not only branches for each device, but also branches for each channel of the device. This allowed the most flexibility for the users who can look at devices or channels. Objects were created for both the fast and slow machine data. Currently, the objects for the slow tree mimick the event record format that the Buffer Manager uses, but the author is certain there is a better way to handle the slow event. However, due to time constraints, this was not explored further.

VI. DATA OUTPUT MODULES

The most important piece of the Event Builder is its output routines. Without these, there would be no data for the experimenters and machine operators. The proposed design for the Event Builder output involves a flexible and dynamically reconfigurable output. This means that as the DAQ is running, there should be the capability to change what formats are being output by the Event Builder. One way to achieve this is to encapsulate all of the work of output in a given format into an object. The sole task of this object is to get data from the Event Builder and correctly output it in the desired format. Then all that is needed is some mechanism to manage these objects.

ASCII Output Module

One of the output formats that was implemented was a simple text output module. The primary purpose of such a format is debugging the DAQ. It is a quick and very readable form for the data which will allow the operators and system designers to ascertain if the system is working as it was intended to. This was not a very complicated object to make, and essentially all of the work for output in a human readable format had been completed during work on the Event Parser. This module, along with the ROOT output module relied heavily on the functionality of the Event Parser to split up the incoming data into its logical pieces.

ROOT Output Module

The purpose of the ROOT output module was to take the event records and fill a ROOT structure as described above. This will be the most important piece of the Event Builder when the machine starts running as this will provide the primary means with which the operators and users will interact with the data. The root files and the associated format will effect everyone using data from the machine. Consequently, much care had to be taken when writing this object. At the end of the time for the project, the object would correctly fill the fast machine data into the appropriate tree. The slow machine tree, at the moment, does not get filled, but will be a relatively trivial addition, and was not completed only due to time constraints.

Output Handler

Managing the various output modules is also done by an object. This object keeps track of all the output modules and guarantees that each completes its output of the events. A multi-threaded approach was taken to this handling. The reason for this was that the writing may take different amounts of time for different modules. Each module is run as its own thread so if one of the modules is waiting for an operation to complete, the other modules can continue working. This allows the usage of time to improve drastically over the non-threaded approach.

With the multi-threaded approach a difficulty arises in synchronizing the various output modules. It would not be good if one of the output threads was still working when the next

event was loaded. This would result in a useless collection of garbled data. To synchronize the output process an additional thread was created to keep in contact with the output modules and assure they have finished their work before retrieving the next event.

VII. CONCLUSION

At the end of the project, each of the components of the Event Builder has a working solution. This will allow the assembly of the Event Builder from these components, and upon its completion, the entire DAQ system can, in principle, be run. Though there are still a few pieces of the DAQ that are missing, the data flow is complete and it will be possible to get data starting at the devices on the beamline into the ROOT files at the other end.

VIII. ACKNOWLEDGEMENTS

The author would like to thank the following individuals: Tim Wilksen for his guidance during the project and for proposing the project, and Rich Galik for organizing and running the program. This work was supported by the National Science Foundation REU grant PHY-0243687 and research cooperative agreement PHY-9809799.

---

[1] "The Conceptual Design Report for the TESLA Test Facility", Edited by D. A. Edwards, DESY, Hamburg, 1995. http://tesla.desy.de/TTFReport/CDR/TTFcdrTab.html

[2] K. Rehlich, "The TESLA Test Facility as a Prototype for the Global Accelerator Network", PCaPAC 2002

[3] G. Gyrgiel, O. Hensler, K. Rehlich, "DOOCS: a Distributed Object Oriented Control System on PC's and Workstations", ICALEPCS 97, Beijing, 1997.

[4] Patrick Fuhrmann, "dCache, the Commodity Cache", Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies

[5] Grid Physics Network Homepage, http://www.griphyn.org/

[6] Rene Brun and Fons Rademakers, "ROOT - An Object Oriented Data Analysis Framework", Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.