Global Accelerator Network

Jeremy Dick

Computer Science, Wayne State University, Detroit, MI, 48202

Abstract

The broad scope of the Global Accelerator Newtork project, proposed to me by Professor Donald Hartil, is, over the next few years, to develop the analytical and hardware interface software of Cornell Electron-positron Storage Ring (CESR) into remotely accessible interfaces such as a web browser application so that is can be remotely monitored and operated. Since there are several security and application issues with the hardware interface idea that have yet to be addressed we are just concentrating on the analytical software for now. Specifically, we are starting with the fault data analytical program of the COMET data digitizer system. This program is referred to as COMET VIEWER and currently runs under x-windows. My goal this summer at Cornell University was to find an alternate method of remotely accessing and viewing the COMET data.

Introduction

At Cornell University, the physics department houses the Cornell Electron-positron Storage Ring (known as CESR) 12 meters below the athletics field and connecting to the lowest level of the Wilson Laboratory. This 250 meter wide ring is capable of producing collisions between electrons and their anti-particles, positrons. When an electron and positron collide and annihilate, the flash of energy results in the creation of new matter, sometimes exotic and unfamiliar. The products of these collisions are studied with a large and complex detection apparatus, called the CLEO detector. To create these collisions, electrons and positrons are accelerated in a linear accelerator (LINAC) to an energy of about 150 MeV. They are injected into the synchrotron and accelerated to 5 GeV before being transferred to the storage ring. Here the particles now wait. Each time this cycle takes place a single bunch of particles is added to the storage ring. Before testing can begin, the storage ring is filled with 5 trains of bunches, with 9 bunches to a train, giving 45 bunches all together. Thus the particles must spend some time in the storage ring being gathered before testing can begin. This is usually where problems occur. There are various factors that can cause the particle beam in the accelerator to be lost. The particles travel in a circular orbit in vacuum under the influence of a magnetic field. Thus if the vacuum is not entire or one of the magnets is malfunctioning, the beam can loss it's course and disappear into the chamber wall. The particles coasting around the storage ring radiate a beam of X-rays, depositing energy on the vacuum chamber wall. This loss of energy causes a loss of acceleration that must be compensated by radio-frequency (RF) cavities used to give the beam an extra boost. Again, a malfunction in one of these cavities can cause the beam to lose acceleration and again be lost into the chamber wall.

Whenever a beam loss system is detected, a system called COMET, which is a network of hardware interfaces and VAX computer systems, takes a snap shot of the status of all the equipment mentioned above that might have cause that event. The most prominent types of data recorded are Beam loss which records horizontal and vertical beam position and current of the first bunch of each of the nine trains of particles in the storage ring, Feedback which is similar to the Beam loss data except that it stores data from almost all of the bunches in each train over a longer time period using an extended memory system, RF which records the RF cavities forward and reflective power and current, and Fast RF which is similar to the RF data except that it records more data over a shorter period of time to take a more precise look at a particular moment of activity.

This data is then digitized into files to later be analyzed by the human eye. At the moment, this data is read and sorted by a program called COMET VIEWER, which then graphs that data onto a computer monitor for much easier interpretation. The want then becomes to be able to do that same thing from outside of the CESR network. The previous method was to telnet into one of the CESR system machines and, with the properly installed software on the remote system, run the VIEWER program and redirect the output to the remote system's screen. This, however, takes a sufficient knowledge of the VMS systems and, again, requires one to know and have the right software to be installed on the remote system. The idea was then to write a version of this VIEWER program specifically designed for remote access. The most popular and effective coding used for remote access programs today is Java, so that seems to most obvious route.

Results

At first, this seemed like a rather easy idea. The current version of VIEWER had been designed with a GUI development kit called X-designer. This package has the ability to generate code for a GUI in multiple anguages such as C++ and UIL. The newest version was reported to now generate Java code, so we acquired this package and planed to just perform a few updates to the VIEWER program and then generate the Java code to use. This however, was before we had a very good idea of how Java operated. So with this plan in mind, I began to learn the X-designer program and then began to update the current VIEWER program. As the old version stood, the layout of the multiple windows used through out the program was a little hard to work with. You had the initial window which is where you pick your data type from a pull down menu, then you have a second window that pops up to chose the file to open which are named according to there date, a third window it then available to pick your data to be plotted, and then a forth and final window with your plots appears. This is fine until you want to plot some new data, then things get a little confusing, having to switch between all these windows. The program is also a little quirky when run through an x-windows emulator, such as Exceed, on a non-VMS system such as one of the Windows based IPAQs, which is what I work in front of. When the graph window is up, any other window moved over it whites out that part of the graph window. Thus, using a new subroutine to run the graph window, I eliminated that white out problem. I then reorganized the data type selection window, the specific data selection window, and the graph window all into one main window for much easier operation.

About this time, I was informed by one of the professors assisting me on this project, John Sikora, that it seemed the version of X-Designer that was supposed to generate our

GUI into JAVA did not perform this function very well. Thus, after making the final touches on the VIEWER program, we moved on to find another way to make our JAVA program. The conclusion we came to after giving this some thought was that we would just have to do it by hand. This, though, proved to be much more complicated then we had planned on. After reading through a JAVA tutorial on the Sun web site, I realized there we multiple ways to accomplish a remotely accessible program. We had the choice between JAVA applets that could be ran within a web browser or the most traditional method of having a client side and server side of an application that talk to on another through sockets associated with specific communication ports on each of the computers. We decided a JAVA applet would be much easier on the user's end, which was the whole point of this project.

We decided the time I had left now was only enough to create a test applet that would at least show what we were trying to do was possible. We were trying to do actually, was to create a JAVA applet that could, when ran from any computer anywhere in the world, have the ability to read in data from the CESR control systems. If this were possible, then a JAVA applet version of the VIEWER program would be quite easy to create. The example I decided to try was to create a JAVA applet that would feed in an array on electron/positron currents, in the form of integers, off one of the CESR control systems and then display them as a bar graph into a drawing area on the JAVA applet. We found, however, that this was not so simple. Most of the programs on the control system and the data collected are coded in Fortran. However, JAVA cannot call Fortran directly since, well, Fortran is just a language I guess JAVA and a lot of newer programming languages are concerned with. To get around this problem, we decided to first create subroutine in C that would feed in the Fortran array and set is equal to an array of it's own. Then we could create a JAVA class file that would call this C subroutine, feed in the C array, and open it for JAVA to use. Then the JAVA applet would simply have to call that JAVA class file in order to use this array. This, however had a problem all it's own. There was plenty of documentation on how to do this by the method appropriate for most platforms such as Windows, Macs, and UNIX. However, the control systems were based on the VMS platform, which required some of these array transfers done by a different method. While John Sikora researched into how to do this, I continued on with learning how to use JAVA to call in an array from a JAVA class file and then plot it onto the screen within an applet. This I accomplished with out much trouble and John found documentation off the COMPAQ web site that discussed how using JAVA with VMS and had created the a JAVA class file that read in a Fortran array through C as mentioned earlier. When we tried to make my applet use his class file, we yet again ran into another problem. Apparently, JAVA applets have certain security restrictions that do not permit it to access sensitive systems such as the CESR control systems. We had hit a block and had come to just about the end of my time here at Cornell so we left the project at that.

Conclusions

The future of this project is to first get past these security restrictions. We believe that it the array is transferred first to another less sensitive environment like a public computer of some sort before it is attempted to be accessed by the JAVA applet, it may work. If this is not possible, then this applet may have to be tried again as client side and server side applications. Once an array of the control systems can be read in, then the final step with this program will be to use it to feed in real time active data from the accelerator. Once this is all accomplished, an actual full scale JAVA version of the VIEWER program can be developed.

Acknowledgments

I am pleased to acknowledge Professor Donald Hartil of Cornell University who proposed this Research Experience for Undergraduates project. I would also like to thank John Sikora for extra guidence through out this project. This work was supported by the National Science Foundation REU grant PHY-0113556 and PHY-9731882 and research grant PHY-9809799. I special thanks to Giovanni Bonvicini and David Cinabro for organizing this program for the Wayne State students.