1.1 The Fermilab Accelerator Control System

K. Cahill, L. Carmichael, D. Finstrom, B. Hendricks, S. Lackey, R. Neswold, D. Nicklaus, J. Patrick, A. Petrov, C. Schumann, J. Smedinghoff, and G. Vogel
Mail to: patrick@fnal.gov
Fermilab, Box 500, Batavia, IL USA 60510

1.1.1 Introduction

For many years the Fermilab physics program has been dominated by the superconducting Tevatron accelerator producing beams for many fixed target and the proton-antiproton colliding beam experiments CDF and D0. More recently, major experiments have used beam from intermediate accelerators. The MiniBooNE and MINOS experiments use 8 and 120 GeV beam respectively for neutrino oscillation studies. Several other experiments and test beams have also used 120 GeV beam. This paper describes the control system for the accelerator complex that was originally developed for the start of Tevatron operation in 1983. This system is common to all accelerators in the chain, and has been successfully evolved to accommodate new hardware, new software platforms, new accelerators, and increasingly complex modes of operation.

1.1.2 Fermilab Accelerator Complex

The Fermilab accelerator complex (Figure 1) consists of a 400 MeV linac, 8 GeV Booster synchrotron, 120 GeV Main Injector, 980 GeV Tevatron based on superconducting magnets, an anti-proton collection facility, and an 8 GeV anti-proton "Recycler" storage ring in the Main Injector tunnel. Beam is delivered to 8 and 120 GeV fixed target experiments, to an anti-proton production and accumulation facility, a high intensity neutrino source, and a 1.96 TeV proton anti-proton collider. The final Tevatron fixed target experiments ended in 2000. In 2001 a Tevatron collider run ("Run II") began with substantial upgrades from the previous 1992-96 run and continues at this time. Prior to that, Fermilab had never mixed collider and fixed target running. However, in late 2001 an 8 GeV fixed target experiment ("MiniBooNE") began operation, followed by 120 GeV fixed target experiments in early 2003, and the NUMI neutrino beam generated from the Main Injector in late 2004. The control system is required to support all these operation modes simultaneously.

1.1.3 Control System Overview

The Fermilab accelerator control system, often referred to as ACNET (Accelerator Network), is a unified system controlling all accelerators in the complex including all technical equipment such as water and cryogenics. ACNET is fundamentally a three tiered system (Figure 2) with front-end, central service, and user console layers. Front-end computers directly communicate with hardware over a wide variety of field buses. User console computers provide the human interface to the system. Central service computers provide general services such as a database, alarms, application management, and front-end support. The central database is a key

component of the system by not only providing general persistent data support, but also by providing all of the information to access and manipulate control system devices. Communication between the various computers is carried out using a connectionless protocol also named ACNET over UDP. The global scope of the control system allows a relatively small operations staff to effectively manage a very large suite of accelerators and associated equipment.

The control system was originally developed for the Tevatron, which began operation in 1983, and applied to all accelerators in the complex at the time. While the fundamental architecture has remained similar, in the years since there has been considerable evolution in field hardware and computing technology employed. This has allowed the system to handle new accelerators and increasingly complex operational demands.

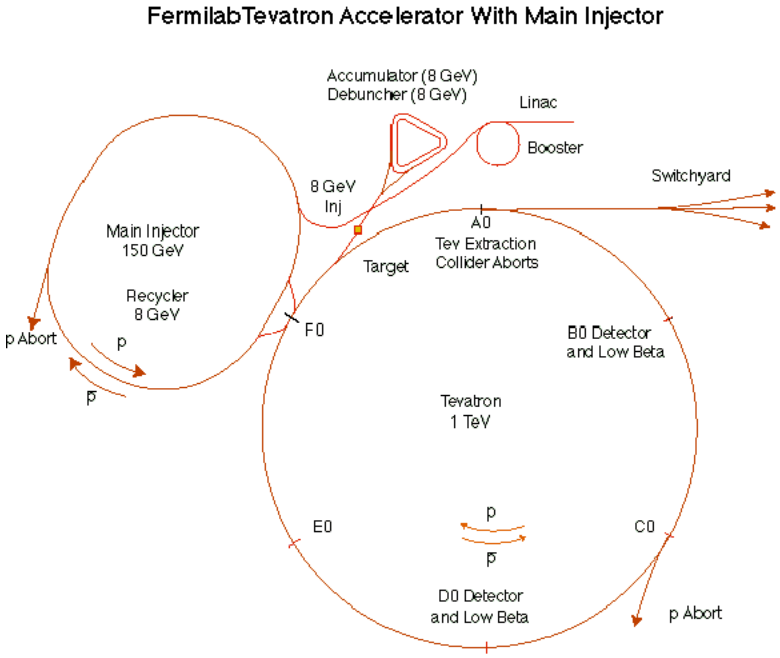

Figure 1: The Fermilab Accelerator Complex

1.1.4 Device Model

Access to the control system follows a device model. The ACNET system employs a flat model with names restricted to 8 characters. While there is no formal naming hierarchy, by convention the first character of the device refers to the machine and the second character is always a ":" For example, T: for Tevatron devices, B: for Booster devices, etc. Informal conventions in the remaining 6 characters provide some categorization by machine subsystems. Recently it has become possible to assign a 64 character alias, allowing a more verbose device name. However this is not yet in wide use. Each device may have one or more of a fixed set of properties including reading, setting, digital status and control, and analog and digital alarm information. Reading and setting properties may be single values or arrays. While mixed type arrays are not transparently supported, this is often done with the required data transformation between platforms done in library code. Device descriptions for the entire system are

stored in a central database. Entries may be created and modified using either a graphical interface or a command language utility known as DABBEL. There are approximately 200,000 devices with 350,000 properties in the Fermilab system.

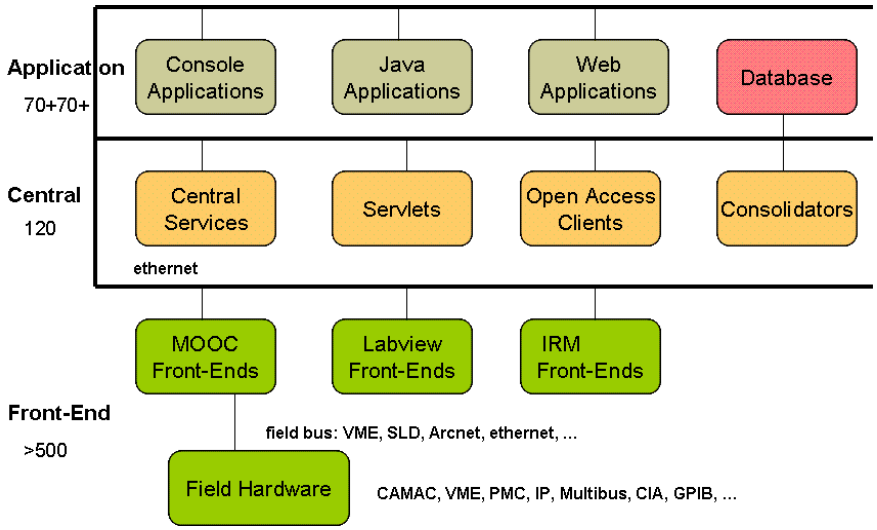

Figure 2: Control System Architecture

1.1.5 Communication Protocol

Communication among the various layers of the control system is done through a home-grown protocol known as ACNET. The ACNET protocol was created in the early 1980s and transferred accelerator data between PDP-11s and VAX systems over DEC's proprietary PCL (Parallel Communication Link.) As the decade progressed and newer hardware became available, IEEE 802.5 token ring and later Ethernet was added to the controls system's network and ACNET adapted accordingly. By the end of the decade, Ethernet and TCP/IP emerged as the dominant network technologies. So, ACNET became a protocol carried over UDP. This move provided three benefits. It allowed any TCP/IP-capable platform to use ACNET. It supported the use of commercial routers to make ACNET packets easily available across WANs, and it let the IP layer handle packeting issues. The ACNET protocol is currently supported on Linux and VxWorks platforms. The Linux implementation is written so that it should be easily portable to other UNIX-like operating systems.

ACNET was designed to be, foremost, a soft, real-time data acquisition protocol. Limitations on network bandwidth and processor memory at the time resulted in a very efficient design for returning machine data at high rates with minimal overhead. As a result, returning large data types can be awkward, but returning lots of small pieces of data (the typical case) works well.

Messages are directed at specific tasks on the target node; a daemon process receives and forwards the messages appropriately. ACNET is a multilevel protocol. At the lowest level ACNET peers communicate by one of two methods: Unsolicited Messages (USMs) and Request/Reply Messages. USMs are less "expensive" than Request/Reply transactions and are useful for broadcasting state to multiple clients.

Request/Reply communication, however, is the main workhorse of the control system. A requesting task sends the request to a replying task. This can either be a

request for a single reading, or a request for multiple periodic readings without re-requesting. A single packet may include requests for data from multiple devices. The replying task then sends one or more replies to the requester asynchronously. If the replier needs to stop the replies (due to an error, for instance), it can include an "End-of-Mult-Reply" status. Likewise, if the requester no longer wants the stream, it sends a cancel message to the replier, which shuts down the stream. Multicast requests have recently been added to the protocol, in which case the requestor will receive streams of data from all repliers in the multicast group.

Higher-level protocols atop the request/reply layer provide the specifics for data acquisition. The primary one is called RETDAT (RETurn DATa) and is used for simple data acquisition. It allows a process to receive a stream of data either at a periodic rate or whenever a specified clock event occurs. The newer GETS32/SETS32 protocol adds a more comprehensive set of event collection options and includes precise timestamps in the reply to aid in correlation of data across the complex. The Fast Time Plot protocol is used for acquiring device readings at high rates. To reduce the required network bandwidth, readings at rates up to 1440 Hz are blocked into single network packets and delivered to the requester several times per second. The Snapshot protocol specifies acquisition of a single block of up to 4096 points of data at whatever rate can be supported by the underlying hardware.

1.1.6 Timing System

ACNET makes use of several types of timing systems to coordinate operations of the accelerator complex. Overall system timing is provided via the timelines that are broadcast on the TCLK clock system. Individual accelerator beam related timing, associated with such devices as kickers and instrumentation, is supplied by the Beam Sync Clock systems. Slowly changing machine data (<720 Hz.) which is useful across the complex (accelerator ramps, machine state settings, beam intensities, etc.) is made available via the MDAT (Machine DATa) system. These timing system signals are available via both hardwire (fiber and copper) transmission and network multicast.

TCLK is the highest level clock system in the complex. It is an 8-bit, modified Manchester encoded clock transmitted on a 10 MHz carrier with start bit and parity. Clock events can be placed no closer than 1.2 μS apart on the hardwire transmission. The network multicast of TCLK provides a 15 Hz transmission of groups of TCLK events that have occurred during the previous 67 msec period. This provides soft real-time information to user applications and central service processes without requiring a special clock receiver card. Timelines are groups of TCLK events that define machine cycles that repeat over a given time period. Timelines are made up of “modules” that define what happens within the complex (with required internal timing and machine state info) over the period specified by the module. A typical timeline module will include TCLK reset events associated with the Booster, Main Injector and the destination machine/experiment. A VME based front-end with special application software known as the Timeline Generator (TLG, described below [1]) provides a flexible user interface that allows operators to manipulate timelines as needed to meet changing operational conditions.

The various accelerator beam sync clocks (TVBS, MIBS and RRBS) are also 8-bit modified Manchester encoded clocks. However, their carriers are sub-harmonics of the given accelerator’s RF frequency (RF/7, ~7.5 MHz). They all carry revolution

markers sourced from the low level RF (LLRF) systems along with beam transfer events synchronized to the revolution markers. This allows precise timing of kickers and instrumentation.

A more recent addition to the timing systems is the use of states. Machine states refer to phases of operation such as proton injection, antiproton injection, acceleration, etc. They are used for example by the LLRF systems to determine what type of RF manipulations will take place within a given machine cycle and when. The Tevatron Low Beta Sequence state is used to change magnet settings for the low beta squeeze and various other state variable transitions are used to trigger data acquisition. To trigger a state transition, any control system task may set a virtual device in a state server, and the transition is then forwarded by multicast or direct communication to other elements of the system. A few selected state values are transmitted on the MDAT network. This allows for timing flexibility beyond the 256 event limit set by TCLK. Also as state values are held in virtual devices, applications may query them at any time.

1.1.7 Supported Hardware

The ACNET control system comprises a variety of hardware and field buses that have evolved over the life time of the accelerator complex. While the functions have remained similar over the years, new hardware technology has been integrated in to the controls system whenever possible given the schedule and budget. The evolution of the ACNET system's hardware mirrors the evolution of controls hardware technology in general.

Early on, the controls system hardware included PDP-11s connected to Lockheed MAC 16 minicomputers with CAMAC as the field bus. Numerous custom CAMAC cards were developed for the timing system, data acquisition and control functions. Two notable and widely used systems, MADCs and ramp generators, are described in more detail below.

1.1.7.1 Data acquisition:

Analog signals are digitized by in house designed multiplexed analog to digital converters (MADCs) which are connected through a CAMAC interface. The MADCs allowed multiple users to sample 14 bit data from up to 128 channels per MADC at a maximum frequency of ~90 KHz for a single user/single channel. Early models allowed 6 simultaneous continuous fast time plot channels and the newer model of CAMAC interfaces allow 16 simultaneous plot channels. Data can be sampled on any clock event, external input or at a programmable rate.

The MADCs have served well for nearly three decades and are now being replaced with HOTLink Rack Monitors (HRMs). A variety of commercial and custom digitizers are used for specialized high rate applications. There are still significant systems that are controlled with CAMAC equipment and thousands of channels are connected through MADCs.

1.1.7.2 Ramp generators:

During the course of operations many power supplies must be ramped. Often they must be ramped in a different manner depending on the type of beam cycle. To satisfy this requirement we have developed flexible ramp generating hardware which

can save sets of tables locally and play the appropriate ramp on specific clock events. This allows different types of beam cycles to be interleaved seamlessly without having to reload ramp tables using higher level software for each cycle.

1.1.7.3 Field buses:

As microprocessor technology progressed, VME and VXI based designs were incorporated into the controls system and processing was distributed. MAC 16s were replaced with VME and VXI front-ends. Eventually VME became the standard control system front-end platform.

Newer power supply controls are most often implemented in Programmable Logic Controllers (PLCs) and many devices come with Ethernet connectivity. The ACNET control system has been interfaced to several popular manufacturer's PLCs.

GPIB and Ethernet connectivity to instrumentation allows for remote diagnostics including oscilloscopes, signal generators, spectrum analyzers, etc.

1.1.8 Front-End Systems

Data from hardware devices enters the Fermilab control system through the front-end computers. These computers are responsible for acquiring the data from the hardware or field bus and responding to the timing system to ensure prompt collection. These closest-to-the-hardware nodes communicate with the rest of the control system using the ACNET protocol. Another important function is to provide a mapping between the central device database and the actual hardware readings and settings. At start-up time, a front-end may have its device settings downloaded from the central database. To implement these common tasks, three different architectures have evolved at Fermilab: MOOC (Minimally Object Oriented Controls), IRM (Internet Rack Monitor) [2], and OAC (Open Access Client) [3].

The IRM software architecture provides 15 Hz hard real time performance to match the pulse rate for the Fermilab linac. It also provides synchronized data collection across all the 15Hz IRM nodes. Custom processing is possible by adding "local applications" to an IRM node. The IRM architecture is built into a standard VME crate providing multiple channels of general-purpose analog and digital I/O. This off-the-shelf I/O capability of the IRM makes it a good choice for many applications with about 185 in use and it is the standard for controls in the linac. The HOTLink Rack Monitor (HRM) [4] provides more analog channels and higher digitization rates in a more modern hardware architecture.

MOOC nodes are also VME-based, built on the vxWorks real-time operating system running on PowerPC based computers. MOOC provides more customization and varieties of acquisition schemes than the IRM. In MOOC, in object-oriented fashion, the developer writes a software class to support a type of device, and then creates an instance of this class for each device. Thus there is great flexibility in device support, while the MOOC framework provides all interactions with the timing system and the ACNET communications. There are roughly 275 MOOC systems in the Tevatron, Main-injector, and anti-proton source. Data is acquired from a variety of field buses, including VME, Arcnet, CAMAC, GPIB, and others.

OACs are front-ends that typically run on centralized server nodes using a framework written in Java, with no special hardware device connections. OACs use the

same communication protocols as other front-ends, but their position in the system gives them easy access to the central database and data from all other front-ends. Access to the timing system is via the above described multicast. Besides providing utility functions such as virtual devices, some typical tasks performed by OACs include:

- Computational combination of data from other front-ends, including database driven algebraic expressions or custom Java code to perform emittance calculations for example.
- Ethernet-based data acquisition, including commercial hardware such as oscilloscopes, PLCs, or custom Ethernet-enabled FPGA devices.
- Process control, including finite state machines, PID loops, and beam trajectory stabilization.

Additionally, the Fermilab data loggers are built on the OAC architecture. There are about 120 OACs plus another 70 data loggers in the system.

Besides these common front-ends, there are also around 25 systems running LabVIEW. They act as front-end nodes in the control system by the inclusion of LabVIEW modules programmed to follow the ACNET communications protocol. LabVIEW front-ends are typically used in instrumentation systems such as wire scanners or synchrotron light systems. GUIs developed in LabVIEW are generally used only by instrumentation experts. With the systems connected as front-end nodes, data is available to standard control system services such as data logging and alarms. Standard applications provide the required subset of the LabVIEW functionality to operators.

These front-end architectures have proven successful in fulfilling the key requirements of data acquisition, timing system response, communications protocol support, and database mapping across the diverse accelerator chain at Fermilab, supporting both legacy hardware and new systems.

1.1.9 Central Services

The central tier of this three tiered control system houses central services. Examples of central service functionality include alarm collection and distribution, data logging, and servlets supporting web applications.

1.1.9.1 Data logging

Logging of accelerator data is done by tasks writing to MySQL databases distributed over 70 central service nodes. Each distributed data logger supports 18 tables having a nominal capacity of 60 devices circularly sharing a 5e8 point data space. A full logger overwrites old points in 1446 days when sampled at a 15 second interval. Overwrite times are shorter or longer dependent on the number of devices in a logger's table and the sampling frequency. More than 50 of these loggers organized by machine or department and log data on periodic rates, TCLK events or software state event transitions. The remaining loggers are reserved for specific diagnostic functions such as TCLK event or software state transitions or device setting modifications. Other loggers provide archives of logged data. So that no data is lost when a specific logger wraps around, each day's data, currently over 5 Gigabytes, is transferred to permanent archives of several terabytes on spinning media.

1.1.9.2 Sequenced Data Acquisition

Sequenced Data Acquisition (SDA) saves scalar, snapshot, and fast time plot data during defined periods of important machine operations. The most complex example is for Tevatron Collider stores, where a typical shot collects about 25K scalars, 1K snapshots, and a few hundred fast time plots with specific collection requirements. The data is used for post-mortem as well as shot to shot analysis to study trends in Collider performance. An extensive suite of tools automatically produces summary information and plots available via the web. This facility is extremely valuable for studying trends in accelerator performance.

1.1.9.3 Save/Restore

Save/Restore services provide for operator initiated saves of the complex for future display or restore. Four times a day, automatic saves are initiated that encompass nearly all addressable devices on operational nodes. Besides providing a backup to operator initiated saves, the big saves expose lurking data acquisition problems, and the data is reflected into loggers that although is sparse provides a quick historical assessment of nearly all the operational channels of the control system.

1.1.9.4 Alarms

Each front-end is responsible for scanning its devices for alarm conditions. When detected, the front-end sends an ACNET message to a central alarm server [5]. The central alarm server supports an alarm protocol for reporting and clearing alarms from the hundreds of alarm reporting nodes and an alarm distribution methodology that includes multicasting of alarm updates to nodes that service hundreds of alarm display clients. This architecture provides excellent scalability with both the number of alarm producers and alarm clients.

1.1.9.5 Front-End download

When a front-end system is rebooted, it needs to know the current setting values and alarm thresholds for its devices. As front-ends do not have easy access to the main database, a front-end download service provides this functionality. A separate setting service keeps the database up to date. Front-ends forward new setting values to this server that then saves them.

1.1.9.6 Accountability

Various central services record considerable information about control system activities. The data acquisition setting routines forward settings to a service that logs settings for accountability providing application access to who, what, when, and where a setting was performed. Also logged are data acquisition jobs initiated by Java programs, data acquisition errors, application usage, CPU utilization by node, and database queries. This information is made available via web based reports.

1.1.9.7 Data Acquisition Engines

Java clients do not communicate directly with front-ends but instead go through Data Acquisition Engines (DAEs). Performing all data acquisition through the central

layer allows more reliable control of security and settings logging, better isolating the front-ends from improper requests. Also the engines perform consolidation of common requests from Java clients across the control system. A single request is made to the front-end and the data are then distributed to all requesting clients. The engines also simulate data acquisition conditions which may not be directly supported by some front-end systems.

1.1.9.8 Servlets

Several Java based servlets provide control system access to web based applications including the parameter page, logger display, and SDA viewer applications. Logged data acquisition errors, CPU utilization, logger fetch, and SQL statement logging are other examples of servlet-provided access to web based displays.

1.1.9.9 Time-Line Generator

The Time Line Generator (TLG) is an ACNET client-server system that generates timelines which place 256 possible events on to the TCLK network and 16 bit states on the MDAT network. The TLG denotes a move away from flat timeline generation to the production of rule-based structured timelines. Each timeline is built on a user application and then executed on the server, a MOOC front-end. A structured timeline is represented by a set of modules and a rule set. A subset of this rule set includes the priority of each module, starting time of each module, the number of repetitions and the end time of each module. Placement of each module is governed by these rules. Each module consists of a set of events, states and its own rule set. These rules govern event and state placement, linkage between different event types and additional actions. The overall result of the timeline is the generation of a set of events and states with specific rules governing how each event and state is placed and how they react to other events and states. The move to rule-based structured timelines has resulted in a great deal of flexibility and robustness being built into the system. Users can now target a specific component of the timeline to be changed by simply swapping modules in and out of the current timeline. The rule set would then allow the timeline events and states to adjust their priority and placement based upon these changing user specifications. This methodology allows supporting very complex modes of operation of the accelerator chain. Furthermore modifications can be made and implemented very efficiently. The TLG system makes it very straightforward to quickly switch from operations that include NuMI/MINOS, pbar production, MiniBooNe and 120 GeV fixed target, to a subset of these or to a Tevatron injection timeline.

1.1.9.10 Experiment communication

Experiments need to obtain accelerator information such as accelerator state, beam intensities and losses. Also it is sometimes useful for the experiments to send information to the accelerator, such as colliding beam luminosities and collision points measured by their detectors. As the experiments do not have direct access to most of the accelerator control system and also have a different programming environment, this is accomplished by a central service communicating via the XML-RPC protocol. APIs for XML-RPC are available in many languages. Each experiment writes its own applications to obtain desired accelerator information and they may also set virtual accelerator devices with experiment information through this service.

1.1.9.11 Databases

The Fermilab control system utilizes several data storage facilities. The predominant data storage is located in several Sybase relational databases that total over 100GB of data. There are separate databases for all device and scaling information for the control system, application data storage, lattice information, Save/Restore, and SDA data for shot analysis. Datalogger data is stored in distributed MySQL databases containing a total of over 7TB of data. A filesharing service is also available to manage shared access to historical data files used by older applications.

1.1.10 Application Frameworks

Applications are written using one of two frameworks. In the CLIB framework, applications are written in C/C++ for the Linux platform. Graphics is based on a custom library on top of basic X-Window calls. A newer framework using the Java language allows for development of applications that can run on any platform and provides a more modern look and feel. Both frameworks provide common functions and a common look and feel within that framework. Both capture all application code required for operations in a CVS repository and provide a place to launch them.

1.1.10.1 CLIB framework

There are roughly 600 Linux-based applications which are used to operate the Fermilab accelerator complex. They are written primarily by people who operate the accelerator including machine physicists, engineers, and operators rather than by individuals from the controls department.

The structure of these applications is rather simple consisting basically of an infinite event processing loop. The basic events are initialization, user interrupt, periodic (15 Hz), and termination. The initialization event occurs once and allows the programmer to set up any initial conditions, and the termination interrupt occurs once at the end of the program to allow for operations such as saving files and cancelling device requests. The periodic interrupt supports updating displays while the user interrupt event allows the program to respond to a user's request. There are other events that are used to a lesser degree including the notification that a global control system state value has changed and the occurrence of a clock event as examples.

To support the writing of these programs, there is a large shared library named CLIB which stands for Console Library. This library contains approximately 1700 entry points which support such topics as data acquisition, user interface, data manipulation, program control, network messaging, error message handling, ACL (Accelerator Command Language) support, and other miscellaneous routines. CLIB is written and maintained by the controls department and is linked at runtime which allows global functionality changes as well as bug fixes to be implemented easily across the entire suite of applications.

In addition to CLIB, there are other smaller libraries which are called user libraries. Many of these libraries are written by controls department personnel while others are written by machine physicists and engineers.

To simplify the creation and modification of applications and user libraries, there is a C/C++ Software Development Environment (SDE). Its first purpose is to allow users to develop application programs without much software development

expertise. Users must know C/C++ but need not understand Makefiles, compiler/linker options, and revision control systems. A second goal is to make sure operational software is never lost. To this end, the SDE automatically places files in a revision control system, in particular CVS. Capturing the entire source in a revision control system allows the SDE to provide a retreat functionality. A retreat of an application program can be done by any user if it turns out in hind-sight that the most recent change(s) are causing problems. The SDE also provides a facility to develop new libraries to avoid code duplication between similar applications. Libraries can be either statically or dynamically linked.

1.1.10.2 Java framework

Besides the CLIB framework, the Fermilab Accelerator Control System includes a newer infrastructure supporting user applications written in Java. These applications can run under Windows, Linux, Solaris, Mac, and FreeBSD platforms on both central nodes and user computers. There are also a number of web applications providing data to the users via the HTTP protocol. The Java infrastructure consists of three major parts: the application framework, the application index, and the building system.

The Java Application Framework [6] facilitates development of standardized control applications by providing an implementation of a uniform Swing look-and-feel and several core services. This includes authentication, logging, printing, screen capture, submission of data to an electronic logbook, and access to application properties in a central repository. Kerberos V5 is used as a common method of authentication for both standalone and web applications, via a customized Kerberos client [7].

The Application Index [8] is a central web-based database of all Java Controls applications and an application that provides for their launching via Java Web Start. For each program, the database provides a URL of a corresponding JNLP file (a standard descriptor understood by the Web Start client). The URLs are combined in a tree, according to the application's fields of use. The Index also supports searching programs by name, description, and author. The JNLP files are generated dynamically upon each request using current information in the database, such as the program's class path, initial and maximum memory heap size, required version of Java Virtual Machine, and others. This allows for changing of runtime parameters quickly from a single place. The Application Index also allows viewing of central logs and statistics on running applications collected by the Application Framework.

Currently all Java code, including locally developed code and third party libraries, is maintained in a single source tree in CVS. Developers write code using their method of choice. A custom Eclipse plug-in is available that simplifies development of accelerator applications. When ready to install new code in the system, developers commit code to CVS and request a new release via a web interface. A building system (which is a set of Perl and Ant scripts on a central server) schedules new builds, checks out relevant modules from the repository, compiles the code, creates and signs jar files, and deploys the binaries. The latest production version of Java Controls is made available through a shared drive on a file server and over the web. The former is used by various server-side processes, such as servlets and OACs, and for development. The latter is mainly for the web-startable client applications. All released jar files (including third-party libraries) are properly signed so that their origin can be verified using the department's public key certificate.

1.1.11 Key Applications

Described below are some of the more important core applications in the system.

1.1.11.1 Parameter Page

The Parameter Page (Figure 3) is a general purpose program which allows display and control of lists of accelerator devices. For each device it displays the device name, descriptive text, reading, setting, alarm limit/status, and digital status. Device settings, alarm limits, and digital control can be modified by users. Groups of devices may be combined into “knobs” that allow correlated changes to be made to members of the group.

Accelerator subsystems have been organized into a hierarchy of persistent parameter device lists called subpages by the subsystem experts and operators. Users can easily add devices to subpages at any time by entering device names.

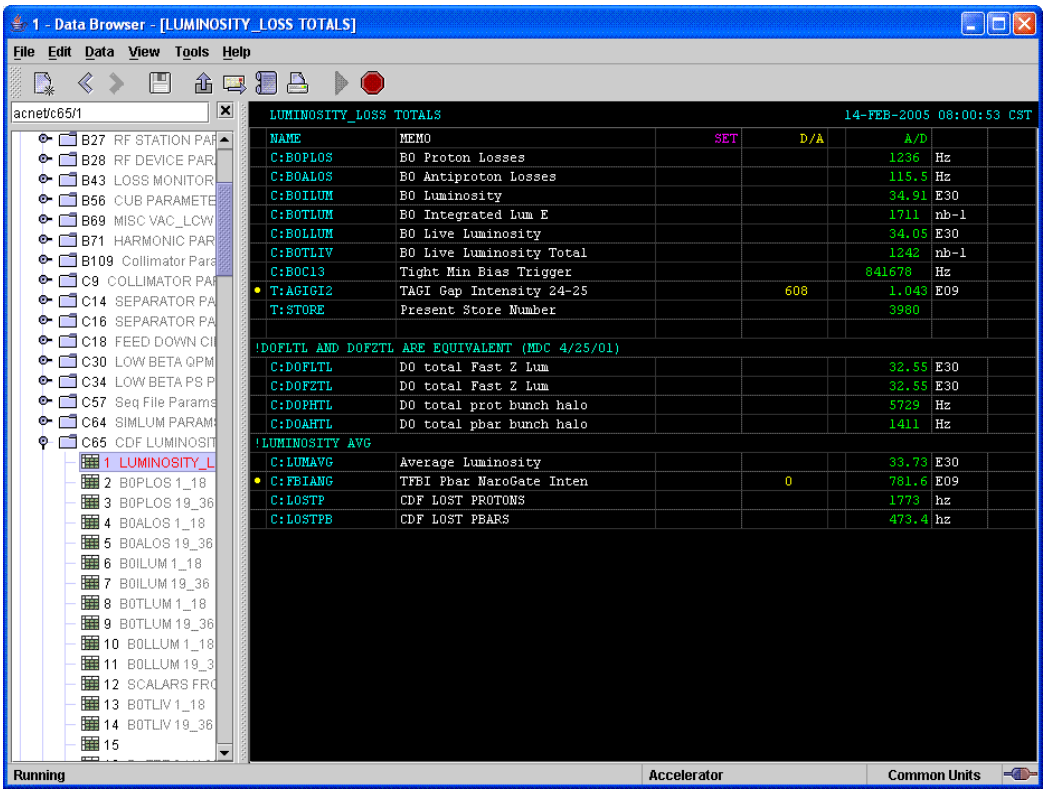

Figure 3: A Parameter Page in the Java Framework

1.1.11.2 Fast Time Plot Utility

The Fast Time Plot utility allows control system users to plot devices in real time. Data sample rates up to 1440 Hz are supported. The x axis can be referenced to any accelerator clock event, time since the plot was started, or another device. Device

readings can be combined into simple expressions using add, subtract, multiply and divide operators before being plotted.

1.1.11.3 Snapshot plot utility

The Snapshot Plot utility supports plots with data collection rates of up to 20 MHz. Hardware determines what rates a given device supports. In addition to displaying snapshot plots for individual users, snapshot plots can be initiated by the SDA utility and the Snapshot Manager for automated data collection and analysis.

1.1.11.4 ACL

ACL (Accelerator Command Language) is a simple to use but powerful scripting language which is designed to operate an accelerator complex. It contains syntax for using control system device specifications in much the same way as variables are used in standard programming languages. There are over 160 commands ranging from controls-specific ones such as read and set to generic if and looping statements. There are also over 100 supported intrinsic functions which can be used in expressions.

The main goal when creating this language was to empower end users to encode powerful algorithms for controlling the accelerator complex. The people who best know what needs to be done to fix a problem or to add a new functionality are often not programmers by nature. They could make up a software specification to solve the problem, but this involves delay and often details are lost in translation. ACL is a solution to this situation.

Machine physicists, engineers, and operators have created many ACL scripts which are executed in the environment of the Fermilab Sequencer. These scripts have helped to make the operation of the accelerator complex more robust and efficient.

ACL scripts can be executed in a number of environments. They can be executed as mentioned above as atomic commands in the Fermilab Sequencer. They can also be executed as embedded commands in parameter pages, and they can be used to update displays and to implement machine control in Lex SA. ACL scripts can be executed directly within any application program using a library routine. In this mode, ACL variables can communicate results to the calling application. There is also a command line interface which is useful for quickly diagnosing problems with the control system.

1.1.11.5 Sequencer

The Fermilab Sequencer [9] is the primary program for coordinating the operation of the accelerator complex. It is especially important for handling the complex sequence of operations necessary for injecting the protons and antiprotons into the Tevatron Collider and bringing them to collision.

The Sequencer was designed so that machine experts could configure it. It could be thought of a simple programming environment using a custom command set consisting of roughly 60 entries. One of these commands also provides an entry into a wider programming environment by executing an ACL (Accelerator Command Language) script. These commands are organized into groups which are referred to as aggregate commands. An aggregate command is usually thought to express the execution of a major accelerator state change. Aggregates are in turn grouped into

modes. Modes generally represent a portion of the accelerator complex such as the Main Injector or the Tevatron Collider.

The Sequencer supports two execution modes, command edit and command execution. The Sequencer is edited by machine experts and operators directly from the program without any formal programming taking place. Once the editing operation is finished, the insertion, modifications, or deletions are stored to a relational database table. There are tables for each individual command type as well as tables for organizing the commands in the proper sequence.

In execution mode, users can choose to execute a single command, a set of commands, or an entire aggregate command. If multiple commands are selected, they are executed sequentially until completion for the most part. There are a few command types which have options for spawning off operations in parallel when sequential operation does not have to be strictly enforced. When an error is encountered, an alarm message is displayed to the user and execution ceases. The user can then decide at what point to resume execution.

Multiple Sequencer modes operate in concert to manage the operation of initiating a store in the Collider. These modes typically coordinate their execution by the use of global control system state values.

1.1.11.6 Lex SA

Lex SA is a user-editable synoptic graphics program based on the CLIB framework. It was created to allow end users to create displays including those which appeared like schematic drawings of various accelerator systems. It supports a set of graphic objects which include drawing primitives, scanned images, and objects which will display the readings of control system values in various ways. There are also objects which can execute ACL (Accelerator Command Language) scripts to change the display or to make device settings. This feature makes the display capabilities extensible by the end user.

Lex SA is actually comprised of two programs, an editor and a display program. In the editor program, users can drag and drop the various types of graphical objects onto the canvas and can edit their properties. Users can also create and save complex graphical objects comprised of primitive objects which they can save away for future use. The entire display can then be saved to a relational database for use by the display program. There is a table for each type of object as well as tables to store how the objects are joined together to make the display.

The display program can be launched from several dedicated applications including the editor. A Lex SA display can also be mapped to each subpage of any parameter page. Once the display is initialized and running, users can click on fields to execute ACL scripts and they can knob analog setting values. There is also a companion program that supports dynamically building a parameter page of devices by clicking objects in the display. There are also objects that when clicked will start up another display allowing individual displays to be linked together.

1.1.11.7 Java Synoptic

Synoptic [10,11] is a client-server system for graphical data representation, similar to Lex SA and EPICS EDM screens. In addition to providing a more modern look and feel than Lex SA, it can display "live" images in conventional web browsers

using very low bandwidth and, in most cases, without the need of additional software. As in the case of Lex SA there are separate display and editor applications.

In a simplest use case, a read-only Synoptic display is opened as a web page (Figure 4) in a regular browser. Upon request, the Synoptic web server makes up a layout of the display, initializes data acquisition, and periodically generates images in a Scalable Vector Graphics (SVG) format. JavaScript code in the browser polls the server every 1-2 seconds for new graphical data and feeds it to an SVG viewer. Since the new image usually looks very similar to the previous one, it is sufficient for the client to receive only differences between successive images. The latest Mozilla Firefox, Apple Safari, and Google Chrome include embedded SVG viewers. Currently Microsoft Internet Explorer requires a third-party plug-in. All listed browsers show SVG images fairly consistently and render partial updates without flickering.

To enable settings from Synoptic to the control system, a display has to be opened in a Synoptic Viewer application. All data acquisition tasks in this case are started locally, and the display's image is rendered directly on the application's canvas. Unlike the web interface, Synoptic Viewer may only be used from certain locations within the lab for security reasons.

A specialized graphical editor called Synoptic Builder is used to create and edit the displays. Like Synoptic Viewer, this is a console Java application, normally launched through Web Start. The builder includes a library of components that can be placed on a display. There are separate components for data acquisition, data transformation, and data visualization. The first two groups are hidden at runtime. The components are interconnected with data pipes. The builder also allows static images and symbols to be included. Synoptic displays are stored in XML format either locally (for later use in Synoptic Viewer) or in a central CVS repository (for both Synoptic Viewer and the web interface).

1.1.11.8 Web applications

A number of applications have been written that run entirely in a web browser. This includes many tools to view SDA data, and a parameter page and device database viewer. Web applications currently do not follow a standard framework. They are written using generic servlets deployed on Tomcat servers with client code written as Java Server Pages (JSPs) or Javascript.

1.1.12 Console Infrastructure

CLIB applications must run under a specialized console environment. This consists of a set of tasks that launch and manage applications, perform data acquisition, handle graphics rendering, and other functions. The console applications framework has evolved from earlier versions of the control system and thus has an older look and feel. Application programs are started from an Index Page program (Figure 5) which displays menus of programs pertaining to each accelerator. Programs have a main window for user interaction and optional windows to display graphics. Applications are run on any of 75 application server computers which contain the complete console framework environment. The framework uses the X Window system for user interaction and display. User workstations can be any internet connected computer which is running an X Window server. Also a special version of the console framework has been created that renders its graphics to a Java applet. This allows running a console from any web

browser without installing special software. For security reasons, only a subset of applications may be launched in this environment and settings are not permitted. A "CLIB Peeker" facility [12] allows one to view internal information about any currently running application.

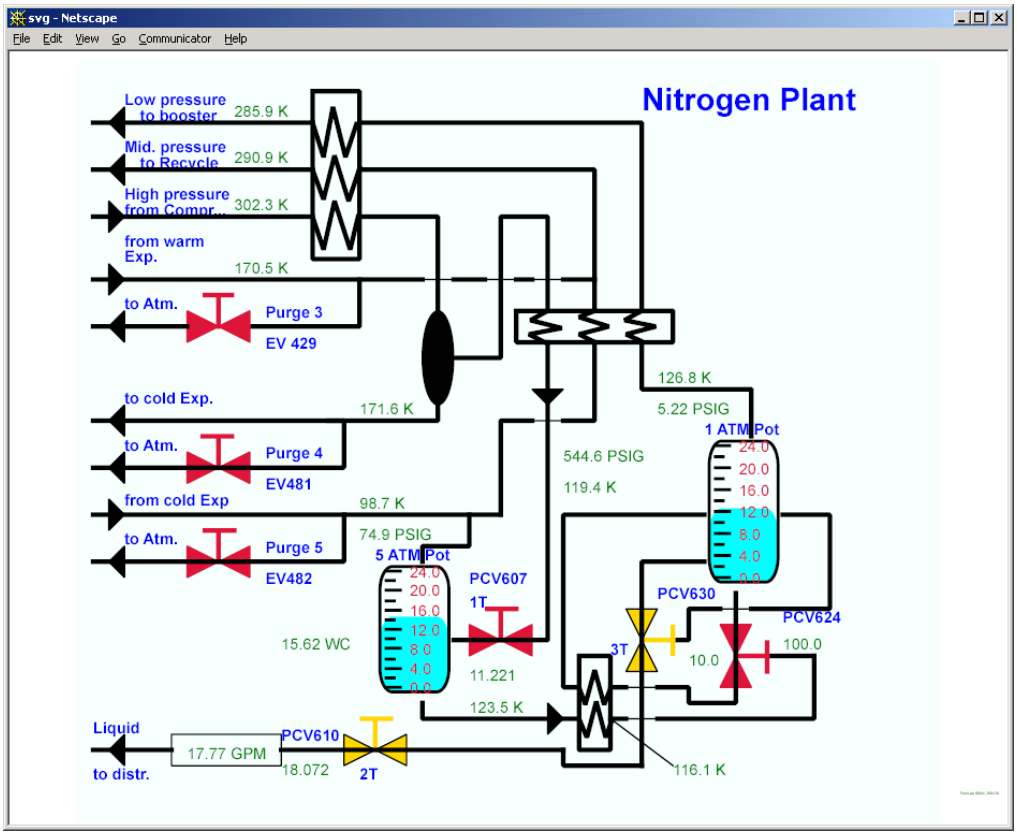

Figure 4: An example synoptic display in a web browser

For Java applications, a separate web-startable Index Page application or Java applet is used. Recently it has become possible to launch Java applications from the above consoles providing a single environment for all applications.

1.1.13 Security

The control system is on a dedicated network inside a firewall that restricts access both in to and out of it. This greatly limits the probability of computer compromise from external attacks.

In such a large and diverse system, it is also desirable to control setting capability even among accelerator personnel. To accomplish this, "classes" are defined and applied to both people and remote consoles. Specific devices can only be set by those in certain classes. Furthermore, applications run from outside the main control room start with settings disabled. They must be manually enabled by the user. These features reduce the probability of accidental settings disrupting operations.

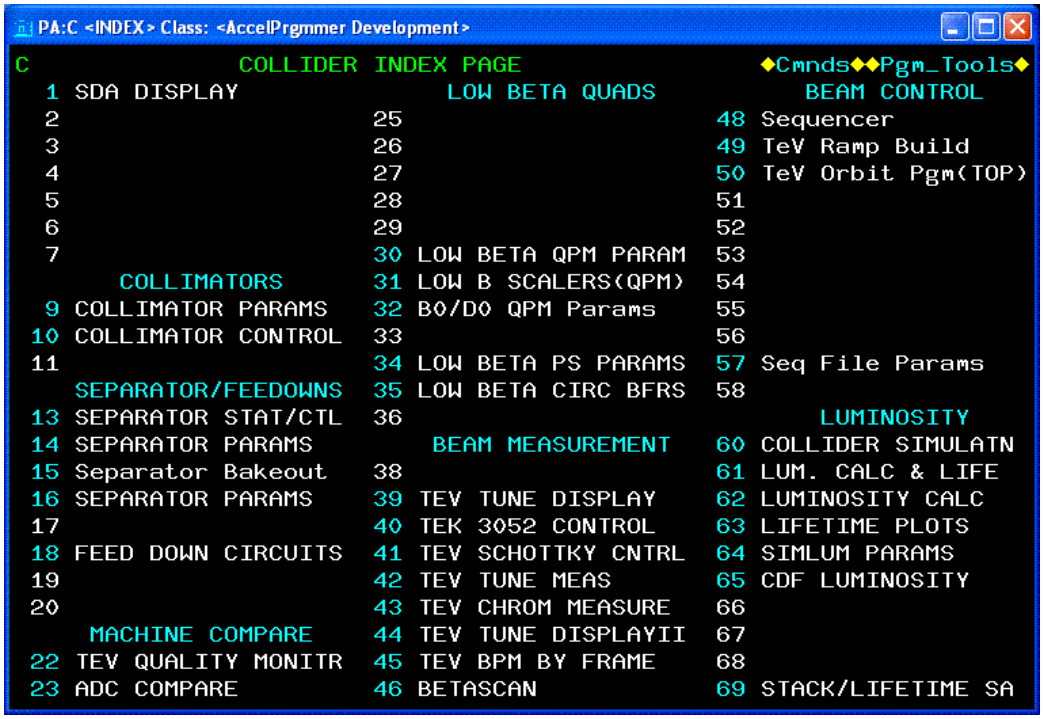

Figure 5: A Console Index Page

1.1.14 Past Evolution

Over the many years since ACNET has been developed enormous advances have been made in computing and hardware technology rendering many components of the original system obsolete. The control system has been able to evolve to take advantage of new technology as well as deal with the increasing operational demands of the complex.

While much field hardware remains in CAMAC, there is now a rich diversity of VME, VXI, Multibus, GPIB, and Ethernet connected hardware. The latter includes commercial scopes and spectrum analyzers as well as custom developed hardware.

Front-end systems were originally PDP-11 and Lockheed-Martin MAC-16 computers. These gave way to i386 Multibus and 68000 VME based systems running the MTOS operating system. All of these older systems have now been replaced by VME based 68040 or Power PC processors running the VxWorks or pSOS operating systems.

Console applications originally ran on PDP-11 computers with custom graphics. These gave way to VAXStations with X-Window graphics. In recent years all VAX software was ported to Linux and now runs on standard PCs.

Communication via the ACNET protocol originally used Digital PCL11-B links. These were migrated to IEEE 802.5 token ring links, and now this traffic travels exclusively over Ethernet.

Programming was originally done in FORTRAN and assembler, and later C. Now C++ is supported, and Java is supported for higher level software. The Sybase relational database was introduced, replacing older VAX based databases.

Hardware subsystems have been continually replaced as needed. Most notably the beam position and beam loss monitor systems for the Tevatron and Main Injector

were replaced in recent years. As these are large systems and there has been limited accelerator shutdown time, these upgrades had to be done in a staged manner during operational periods. Partial old and new systems had to coexist until the replacement was complete.

1.1.15 Future Directions

The Tevatron Collider is currently scheduled to end operation by October, 2010. The complex will then be upgraded to increase the intensity delivered to the neutrino physics program. The new NOνA experiment is expected to run until at least the late 2010's, and other new experiments are expected to make use of 8 GeV beam. The ACNET control system will continue to be used during this era. Obsolete hardware will be replaced as needed, and some effort will go toward modernizing the CLIB application environment.

An 8 GeV superconducting linac, known as Project X [13], has been proposed to further increase the beam intensity available for these and other new experiments. Prototype accelerators, HINS [14] and NML[15], for the Project X linac are currently under development at Fermilab. Until recently they have used the EPICS [16] and DESY DOOCS [17] control systems and been independent of the main ACNET system. A number of front-ends and synoptic display screens have been developed. Effort is currently underway to integrate these facilities back into ACNET. A prototype integration of an EPICS IOC under MOOC has been developed in the style of the EPICS2TINE interface [18]. EDM has been extended to support communication via the ACNET protocol as well as EPICS Channel Access. This allows inclusion of ACNET devices on already developed screens. EDM screens may now be launched from ACNET consoles. Clock events from these facilities have been added to the general TCLK clock event multicast and are available to data loggers and other central services. The current plan is to base the core Project X control system on ACNET, while supporting EPICS IOCs as needed. This will allow other EPICS based labs developing subsystems for Project X to work in the system with which they are most familiar.

1.1.16 Summary

Though originally developed in 1983, the ACNET control system has evolved to meet the increasingly complex operational needs of the Fermilab accelerator chain as well as take advantage of new technology developed since then. Its very modular nature has allowed both hardware and software systems to be upgraded as needed with minimal disruption to operations. The very efficient ACNET communication protocol has handled the continually increasing number of computers and associated volume of data. Solid development frameworks at all levels of the system have met the needs of developers and promoted commonality in the code. Powerful yet straightforward to use core applications such as the Parameter Page, plotting programs, Sequencer, and Accelerator Command Language make the system very accessible to operations personnel. Extensive logging of accelerator data as well as events, settings, errors etc. greatly aid diagnosis of subtle problems in this very large system. With the recent port of all application software to Linux from VAX/VMS, ACNET will be viable for the projected lifetime of the upgraded neutrino physics program and should form a strong basis for the control system of the proposed Project X accelerator.

1.1.17 Acknowledgements

The Fermilab control system has been developed by numerous people over many years. This includes not only controls department members, but machine specialists, operations and other technical support personnel as well.

1.1.18 References

1. L. Carmichael, “Automated Task Scheduling Using Multiple FSMs at Fermilab”, proceedings of the 1997 ICALEPCS, Beijing, China (1997).
2. R. Goodwin, M. Kucera, and M. Shea, “Use of Small Stand-alone Internet Nodes as a Distributed Control System”, proceedings of the 1993 ICALEPCS, Berlin, Germany (1993).
3. D. Nicklaus, “Java-based Open Access Front Ends in the Fermilab Controls System”, proceedings of the 2003 ICALEPCS, Gyeongju, Korea (2003).
4. A. R. Franck, R. W. Goodwin, P. A. Kasley, M. Shea, “HOTLink Rack Monitor”, proceedings of the 2001 ICALEPCS, SLAC, Stanford, CA (2001).
5. S. Ahn, “Fermilab Beams Division Alarms Processing System”, proceedings of the 1999 ICALEPCS, Trieste, Italy (1999).
6. http://www-bd.fnal.gov/controls/java/framework/af-guide.pdf
7. A. D. Petrov and D. J. Nicklaus, “Secure Client Tier for the Accelerator Control System”, proceedings of the 2005 ICALEPCS, Geneva, Switzerland (2005).
8. http://www-bd.fnal.gov/appix
9. T.B. Bolshakov, A.D. Petrov, S.L. Lackey, “Synoptic Display - A Client-Server System for Graphical Data Representation”, proceedings of the 2003 ICALEPCS, Gyeongju, Korea (2003).
10. http://synoptic.fnal.gov
11. J. Annala, “The Fermilab Sequencer used in Collider Operation”, proceedings of the 1995 ICALEPCS, Chicago, IL (1995).
12. J. Wang and B. Hendricks, “A Diagnostic Tool for Console Applications of the Fermilab Accelerator Control System”, proceedings of the 1997 ICALEPCS, Beijing, China (1997).
13. S. Nagaitsev, “Fermilab’s Project X”, proceedings of the XXIV Linear Accelerator Conference, Victoria, B.C. Canada (2008).
14. R. Webber, “Overview of the High Intensity Neutrino Source Linac R&D Program at Fermilab”, proceedings of the XXIV Linear Accelerator Conference, Victoria, B.C. Canada (2008).
15. B. Chase, M. Votava, M. Wendt, “Controls, LLRF and instrumentation systems for ILC test facilities at Fermilab”, proceedings of the 2007 Particle Accelerator Conference, Albuquerque, N.M. (2007).
16. http://www.aps.anl.gov/epics
17. http://doocs.desy.de
18. P. Duval *et. al.*, “The Babylonization of Control Systems”, proceedings of the 2003 ICALEPCS, Gyeongju, Korea (2003).