Gathering Data from the Fermilab Linac Using Object-Oriented Methodology

Elliott McCrory
Fermilab
Batavia, IL 60510 USA

Abstract. For a number of years, a simple set of objects in C++ has been available to the Fermilab Linac Group for accessing data from the Linac control system. This suite of classes is a simple and powerful way to access this system. The objects are based on the way in which the accelerator data are stored in the local control stations and on the protocol through which these data are transmitted on the network. This paper describes the objects and some of the ways in which they have been used. In particular, several multi-purpose UNIX-style data acquisition tools have been written, along with an interface to pre-existing software packages.

1. Introduction

In order to fully describe this simple set of objects, it is necessary to understand a bit of the environment in which this system operates. First, we will briefly describe the Fermilab 400 MeV Linac, followed by a short description of the control system which this accelerator uses. Then, we get into the description of the objects used on the UNIX consoles for this system: abstraction and encapsulation of the data structures and methods. A few of the applications are described, followed by some operational considerations of this system.

The genesis of this work came under Michael Allen[1]. He performed the basic abstraction and encapsulation of the objects described here.

2. The Fermilab Linac

The Fermilab Linac accelerates negative hydrogen ions (H^-) from the ion source to 400 MeV through multiple stages of acceleration: ion source, 750 keV column, 116 MeV 201 MHz drift-tube linac and 400 MeV 805 MHz side-coupled-cavity linac[2]. There are a pair of (redundant) ion sources, five 201 MHz rf systems, eleven 805 MHz systems, a sub-system for the quadrupole magnets in the 805 MHz portion of the linac and a sub-system for the beam diagnostics. Vacuum is controlled, logically, through the rf sub-systems. This linac cycles at 15 Hz and the ion beam can be accelerated on every rf cycle. The beam pulse length varies from 10 to 60 microseconds. Our average current today is 45 mA, for a total delivered charge of as high as 1.7×10^{13} particles per pulse. With minor modification, this charge could be increased to 3×10^{13} ppp[3].

3. The Linac Control System

The control system for this linac is described in detail elsewhere[4]. The primary design goal of this system (in 1981) was to ensure that if a device went out of tolerance, then beam could be disallowed before the next 15 Hz cycle occurred. In the days before the present control system, beam pipes and vacuum valves had been destroyed by the linac beam[5]. Directly from this primary specification falls the need to have the control system rigidly synchronous to the 15 Hz cycle time. The alarm scan for each linac control station happens at 15 Hz, and about 5% of the analog readings in the linac have been enabled to inhibit beam when they go out of tolerance.

The software architecture[6] on each linac control station is loosely coupled to PSOS. Communication from a console computer is established through UDP/IP sockets, using either the Fermilab Accelerator Controls NETwork (ACNET) protocol[7] or through a custom "Classic Protocol."

The linac control system is distributed to seventeen VME-based crates containing MC68020 processors (which could be upgraded to 68040's), communicating with each other and with the outside world via Token Ring. Each of the seventeen stations controls up to six Smart Rack Monitors (SRM) [8], which contain the D/A's, A/D's and digital I/O necessary to talk to real equipment. The VME crate can also contain digitizers, but the only digitizers presently used on our VME stations are 1-to-10 MHz "quick digitizers" [9] for looking at the transient beam diagnostic signals. Each local control station owns approximately 400 scalar, analog devices.

We have recently developed an "Internet Rack Monitor" which is a stand-alone rack-mounted chassis which contains a MC68040/Ethernet/Industry Pack VME card, in addition to the D/A, A/D and digital I/O of an SRM [10].

This control system is used at other locations: the Fermilab D0 experiment, TESLA, Michigan State nuclear accelerator, Fermilab Booster and Main Ring High-Level rf, the Loma Linda Cancer Treatment Facility and the Shreveport PET-isotope production facility.

4. UNIX Data Acquisition and Control

The data acquisition and control software for the Linac consoles runs on Sun SPARCstation computers running Solaris 2.4. It has, in the past, run on SunOS 4.1, SunOS 3 and on the 68020-based MassComp computers of 1988. The system is implemented in C++. The ideas presented here are evolving as the C++ definition evolves and as our experience with objects grows. A FORTRAN interface is maintained, minimally.

5. Abstraction of Data and Operations

The objects in this OO system are abstractions of the data types which are present in the local control stations. These data types are:

Scalar, analog data, the associated binary status and control alarms information and database information associated with a real or derived local device;

Binary data, it's alarm and database information;

VME memory, including vectorized analog data associated with the quick digitizers;

Data streams (a generic way to assemble structures of data in the local station);

The means for gathering these data are encapsulated into the objects's methods. These methods include the network protocol for getting the data, database name resolution for analog devices, synchronized data return at a given rate (up to 15 Hz) and synchronized return of data when a specific Tevatron Clock (TCLK) event is received.

6. Description of the Objects

6.1 Major Objects

The major objects used in this system are described here. Refer to Figure 1 as a guide.

`TRAccessObject`: Parent object of all the data acquisition objects. A closely-related class, `CtlMsg`, handles the information necessary to insure that each object gets the proper information from the network.

`BinaryDatum`: Handles the retrieval and processing of the local station's binary data.

`RemoteMemory`: Handles the retrieval and processing of a local station's VME memory.

`DataStream`: Handles the retrieval and processing of the local station's data stream information.

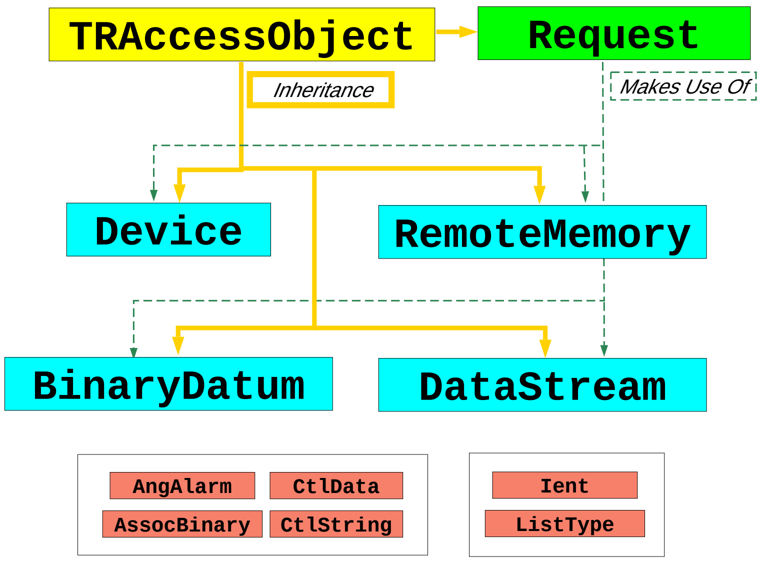

Figure 1, Simple view of the interrelationship of the objects.

`Device`: Handles the retrieval and processing of the local station's scalar analog devices. This class includes: instances of the `CtlData` class for handling the scaling factors on the readings and settings of the device; an instance of the `AngAlarm` class for determining the alarms status of the local device and a copy of the local database information (which can be modified and downloaded to the local station, if desired). Options on this object include: Setting, motor control and associated binary status and control (through `AssocBinary`).

`Request`: Handles the synchronized, repetitive reception of data from the local control stations. This repetitive return is initiated by a single network message. This class includes: Lists of `Device`, `RemoteMemory`, `BinaryDatum` or `DataStream` through the list template `PList<`*`object`*`>`; an instance of `PList<ListType>` to define the type of data returned on the request; an integer representing the period of the return data (1=15 Hz; 30=0.5 Hz, for example); and, optionally, a TCLK event for returning data only on that event. The control of when to return data is handled exclusively by the local station.

6.2 Minor Objects

The minor objects used in this system include:

`ChanIdent` and `AddrIdent`: The logical address of the information within a local control station and on the network.

`AngAlarm`: Handles the analog alarm information for a Device.

`AssocBinary`: Handles the associated binary status and control for a Device.

`CtlData`: Handles the conversion of internal 16-bit data to voltages and to engineering units.

7. Some Applications

We have written a few dozen applications on these objects, and a few of these are summarized in Table 1.

8. Operational Considerations

These classes were derived through the effort of M. Allen in 1988 for the Loma Linda Medical Accelerator, under development at that time at Fermilab. The author has expanded and enhanced these object gradually over the years. R. Florian has contributed a significant number of application programs over these years. In summary, there has been only a minimal effort put into this fairly capable system, no more than 1 person-year.

We recently tested the robustness and throughput of the system on a SPARCstation 2 computer from Sun. This computer was able to flawlessly capture and display over 500 network frames per second using this system.

Application	Description
ac-get-data	General data acquisition on the UNIX Command Line
	Correlation Plots, gating, triggering, and ~ 30 other options
DataViews	Data acquisition interface to the commercial product DataViews
Synoptics	Several synoptic displays implemented using DataViews
checklist	A suite of ~10 shell scripts which runs each day to inform staff of any unusual situation in the Linac control system
	For example: check system date on each local control station, check that some of the important devices acutally are in the alarm scan, check for some rare fault conditions, etc.
Plot package	Using TCL/TK/BLT, scalar and array plot packages
page-g	I/O with the 40x20 dumb-terminal used to access configuration parameters for each local station
IP Node Test	Test that all IRMs are on the network
RDATA Edit	Edit the tables in the local control stations which control the operation of these systems.
Save Restore Data	Edit the analog data description tables in the local control stations.

Table 1, List of some of the applications used in the Fermilab Linac

This system is fully multi-user and multi-tasking.

9. Conclusions

Using an object-oriented approach to data acquisition in the Fermilab Linac has been a direct, simple and powerful way to encapsulate the complexity of data acquisition for this accelerator. New ideas will be implemented as the C++ definition changes and as we become more familiar with them. In particular, templates have not been adequately exploited, multiple inheritance is not used and no polymorphisms have been necessary. This system is evolving.

10. References

[1] Present address: Motorola Corporation, Arlington Heights, IL, email: allen@cig.mot.com

[2] E. McCrory, "The Commissioning and Initial Operation of the Fermilab 400 MeV Linac," Proceedings of the 1994 Linac Conference (Tsukuba, Japan), pp 36-40.

[3] M. Popovic, et al., "Possible Upgrades to the 400 MeV Linac," Internal note.

[4] E. McCrory, R. Goodwin, M. Shea, "Upgrading the Fermilab Linac Control System," Proceedings of the 1990 Linac Conference (Albuquerque), pp 474-476; `http://adwww.fnal.gov/LINAC/linac.html`

[5] Private communications with Charles Schmidt, Linac Group leader.

[6] `http://adwww.fnal.gov/LINAC/software/locsys/locsys.html`

[7] C. Briegel, et al., "The Fermilab ACNET Upgrade," NIM A293 (1990), 235-238.

[8] `http://adwww.fnal.gov/LINAC/hardware/srm/srm.html`

[9] `http://adwww.fnal.gov/LINAC/hardware/quickdig/QuickDigitizerDoc.html`

[10] `http://adwww.fnal.gov/LINAC/hardware/irm/irm.html`