BEAM VIEWER CONTROLS AT JEFFERSON LAB *

M. Johnson†, Jefferson Lab, Newport News, VA, 23606

Abstract

The beam viewer system at Jefferson Lab provides operators and beam physicists with qualitative and quantitative information on the transverse electron beam properties. There are over 140 beam viewers installed on the 12GeV CEBAF accelerator. This paper describes an upgrade consisting of replacing the EPICS based system tasked with managing all viewers with a mixed system utilizing EPICS and high level software. Most devices, particularly the beam viewers, cannot be safely inserted into the beam line during high-current beam operations. Software is partly responsible for protecting the machine from untimely insertions. The variety and number of beam-blocking and beam-vulnerable devices motivate us to try a data-driven approach. A single program reads in configuration data describing the type, position, and interaction with beam of every device. Management and protection is achieved twofold, using EPICS records and feedback loops. This paper will describe in detail the software application, EPICS database templates and C++ servers, and the information flow. In addition, improvements in accommodating hardware failure will be described.

VIEWER SYSTEM

The operation of the CEBAF accelerator at Jefferson Lab relies on a wide variety of devices which must be physically inserted into the beam line when needed. Beam viewers measure beam profile and provide information about transverse beam characteristics. Faraday cups measure beam intensity at several points around the accelerator. Insertable dumps provide device protection and control of beam transport to the experimental end stations. These devices and a variety of others are critical to the delivery of a precisely tuned electron beam.

One software system, historically dubbed the Viewer System, is responsible for handling all requests to insert or retract devices, coordinate device activity, and protect devices from damage due to beam intensity.

The Viewer System has been in place since the 1980s, before commissioning of CEBAF. It has migrated from TACL to EPICS [1], where it ran on 20 Input/Output Controllers (IOCs) spread throughout the accelerator. Coordination and centralized control was achieved through a single state machine running on a critical IOC.

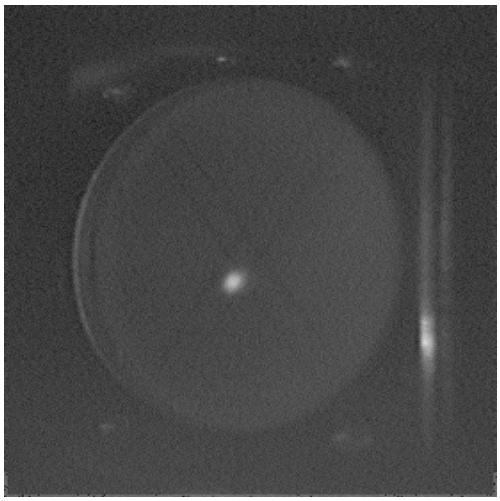

Figure 1: Beam Spot on Viewer

The Viewer System included some 9K lines of State Notation Language code, 2K lines of C code supporting Subroutine Records, and hundreds of configuration files for building the EPICS database. Device configuration data was spread throughout hundreds of files in dozens of folders. To support aggregation of several viewers onto a single I/O card, channel/device patterns were established and followed. For example, several functions of one device would be handled by a heterogenous set of I/O cards by dedicating the first two channels of each card to that device. Unused channels were simply left unused and could not be exploited for other purposes.

Under this architecture, no single entity had complete knowledge of any beam device. For example, a device which has ceased operating would be detected by low-level EPICS record processing chains, and the beam current would be limited until proper operation was restored. Unfortunately, only an examination of each of these records would identify the faulty part. Hardware failures happen often enough that significant time was lost while operators searched through data for signs of a dead device.

Maintenance and updates to this system required considerable effort. Adding or updating a single device required making changes to many database configuration files and to the state machine. Although EPICS provides some state machine diagnostic tools, debugging a large, complex State Notation Language (SNL) program running on an active IOC is challenging, making debugging a challenge.

System behavior as a whole is similarly resistant to

† michaelj@jlab.org

change. The uniform treatment of insertable devices in EPICS code results in protection of all devices from high current beam at all times, which is usually desirable. However, one can envision scenarios and specify conditions where certain devices do not need this protection. For example, and inactive experimental end station may need to reboot an IOC or insert a device as part of a hardware test. Such activities should not limit the beam supplied to active end stations.

This kind of finely-tuned beam-line segmentation of device protection, and the maintenance issues outlined above, motivate us to start over with new software and a new design.

In the following sections, we describe a new data-driven approach to address these problems while providing the necessary device protection and meeting all other requirements. Device configuration data is organized and consolidated, easing the adoption of hardware changes into the software. The static EPICS database has been greatly simplified using macros and templates to create the dynamic database. Software functions have been separated into configuration management, device coordination, and communication with hardware through EPICS Channel Access. We describe the implementation of these modules and some of the classes that support them.

Finally, we describe the new features provided by this system and list some of the planned improvements being developed.

DESIGN

Our new system, dubbed the Insertables System, manages all insertable devices. While these insertable devices vary in their function and impact on operations and experiments, many share common traits and functions. A device needs one or more of several hardware channels : digital output to drive insertion/retraction, limit switch inputs to read position, and video routing channels. At any one moment, a device may be inserted, retracted, traveling, or lost. Lost devices are those which have not responded to commands for a short time.

Although separate software systems manage other functions of these devices, such as monitoring current or water flow, our software system is only concerned with managing the insertion, retraction, and protection of these devices from beam. This idea lends itself to a cohesive design that makes it easier to describe rules for device behavior and to create a system that adheres to those rules.

Configuration

A data-driven approach greatly improves maintainability of the system as a whole. To achieve this, we consolidate all device and channel information into a single, structured XML file. All information about a device, its classification, and its channels can be found within a few lines of configuration data.

Collecting data in this manner also helps with validation of the system as a whole. Hardware engineers, with this data in hand, can find spare channels and duplicate entries. Because the data format is open to any device/hardware mapping, devices can be connected to any channels with no restrictions.

```
<device name="ITV0I02" ioc="iocin3" type="viewer">
<insert_limsw card_name="IN01S19" channel="9"/>
<retract_limsw card_name="IN01S19" channel="10"/>
<solenoid card_name="IN01S18" channel="5"/>
<camera card_name="IN01S21" channel="5"/>
<light card_name="IN01S18" channel="16"/>
</device>
```

Figure 2: XML Configuration Data

The problem now becomes one of parsing this configuration data and creating a collection of objects corresponding to the entities described.

EPICS Database

The EPICS database now has limited duties.

Low-level hardware records carry hardware signal data. A dozen or so records are needed to support a device and give its status and characteristics. One file now supports all EPICS records related to one device. EPICS macros are used as a placeholder for device names and channel information. The static database is very simple, and must be loaded once for every device present in the system to create the full dynamic database needed to support live devices. This reduction of database code makes debugging and feature upgrades easier.

Control of the beam current is taken out of the EPICS database for simplicity, leaving it to a high level application to control. One consequence of this design choice is that the IOC relies on the high level application to run reliably. As a failsafe, an IOC in the Insertables System will limit the beam current if a regular heartbeat single is not maintained by the high level application.

Modules

A single command-line program, the Insertables Manager, is responsible for consuming this configuration data, deriving the relationships between entities, and managing the results.

The Insertables Manager can be made to produce a table of hardware channels. With a different set of options, it produces a set of dbLoadTemplate commands for IOCs to use at boot time. These dbLoadTemplate commands are supplied with macro values for each device. Finally, the Insertables Manager starts the background task operation of coordinating all device insertion and retraction activity.

The Coordinator module runs in the background and monitors all EPICS channels through an event loop. The Coordinator has the flexibility to deal with changes to device state as soon as they happen. Having a centralized coordinator facilitates better communication between devices.

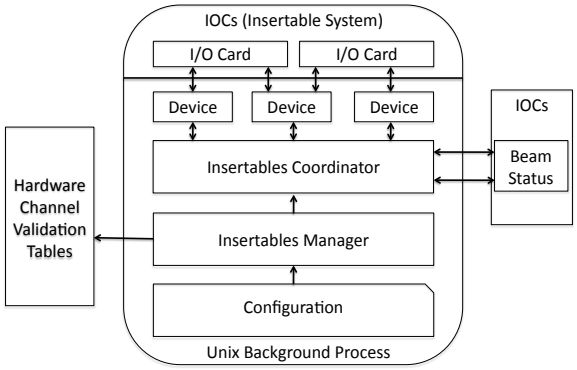

Figure 3: Insertables System Diagram

```
// - Dumps
std::set<Device*> m_dumps_retracted;
std::set<Device*> m_dumps_insert_waiting_for_ok;
std::set<Device*> m_dumps_inserted;
std::set<Device*> m_dumps_retract_waiting_for_ok;
std::set<Device*> m_dumps_lost;
std::set<Device*> m_dumps_hardware_protected;
// - Standard Viewers
std::set<Device*> m_viewers_retracted;
std::set<Device*> m_viewers_insert_waiting_for_ok;
std::set<Device*> m_viewers_inserted;
std::set<Device*> m_viewers_retract_waiting_for_ok;
std::set<Device*> m_viewers_lost;
std::set<Device*> m_viewers_hardware_protected;
```

Figure 4: Device Collections

Class Design

The Device class maps hardware inputs to devices and collects status information based on hardware feedback. This class tracks the static configuration and dynamic status of every device. Device configuration includes data about whether the device is vulnerable to high current beam or impervious to it, as well as the names of the controlling IOC and rack where it is located. Device status is an enumerated list of values corresponding to the possible states of the device.

The Device class hides these internal details and exposes commands to insert or retract a device, route its video, or check its status or configuration.

All hardware values are monitored using EPICS records, or Process Variables (PVs). The EpicsPv class provides a simplified interface to the EPICS Channel Access (CA) [2] system. It provides Channel Access subscriptions through a factory method, and supports an observer framework so that clients can be notified of changes to PV values.

The job of the Coordinator class is to tie all of these elements together, respond to hardware input and user commands, and actuate devices accordingly. With a global view of all elements, the Coordinator class can make decisions based on the state of the whole system.

IMPLEMENTATION

Device objects are organized into C++ Collections (sets, maps, vectors). A Device responds to events based on its membership in a given collection.

For example, if any device belongs to the __lost group, a message will be generated which limits the beam current while engineers resolve the hardware issue. A signal from the device's limit switch can cause the Device object to be removed from this set and restated to the __inserted or __retracted sets.

In another example, inserting a beam viewer needs to generate retraction signals for all other inline devices, to get a clear view of the beam spot. This is achieved by checking the __inserted sets for members, and sending retraction signals to all of them.

This architecture allows us to be very flexible in response to events, and to improve the algorithms later with only small code changes.

EPICS Database and Templates

The EPICS database has been simplified to a few macro-driven templates. A template for a Faraday Cup, for instance, will have records reflecting its position (inserted, retracted, or moving). A template for a beam viewer would have additional channel connections for video routing.

C++ Code

Procedural code contains almost no static knowledge of individual devices. The configuration data drives the creation of Device objects, and hardware events and user commands result in actions that move those objects in and out of appropriate collections. Code is reserved for device behavior, while properties and configuration of actual hardware is left for the configuration data.

CONCLUSION

The Insertables System now serves as our replacement for the Viewers System. Safety protocols have become more sophisticated. When experimental end stations are being protected by beam stopper devices, the Insertables System will not try to protect devices in those end stations using software. This has the effect of reducing the amount of time we spend limiting beam current to experiments when others are reconfiguring.

Removing the monolithic state machine from an IOC and replacing it with a C++ program running on a UNIX host takes away a critical piece of code from a frequently-rebooted IOC. The server process should now run for weeks at a time without restarting.

Static configuration data is more readily accessible. Our system generates hardware-centric configuration tables for hardware and software engineers to use as collaboration

tools. Verbose dynamic status information is also available in the system. Each device now supports a status string detailing its last known state. Faulty limit switches and other hardware can be found more easily.

Future Work

This new system is well positioned to maintain and enhance. The modular design allows us to quickly implement upgrades once they are envisioned by accelerator scientists and operators. More sophisticated beam current limiting rules are being explored. Beam stopping devices and laser configuration are being considered as criteria for device protection.

New devices can be supported with relatively few changes. A new device type is easily added to the configuration data, and the Coordinator module dynamically adds new devices to appropriate sets and monitors their behavior. More device types may be added to this system in the future, allowing us to remove these duties from other systems and reduce overall code complexity in the facility.

The semistructured XML configuration file largely echoes data already found in the CEBAF Element Database (CED) [3]. The XML parser currently used will be replaced with connection to CED's SQL Server Database. Using this method, a single unified view of the hardware configuration will be shared by all groups and systems, reducing errors and quickly propagating changes throughout the organization.

ACKNOWLEDGEMENTS

Many thanks are owed to many people in the Instrumentation and Control group, the Accelerator Operations group, and the Controls Software Group. Special thanks go to Alicia Hofler for her ceaseless help and explanations.

REFERENCES

[1] S. A. Lewis, "Overview of the Experimental Physics and Industrial Control System: EPICS," (2000) http://csg.lbl.gov/EPICS/OverView.pdf

[2] J. O. Hill and R. Lange, "EPICS R3.14 Channel Access Reference Manual," (2013) http://www.aps.anl.gov/epics/base/R3-14/12-docs/CAref.html

[3] T. Larrieu, M. Joyce, C. Slominski,, "Design and Implementation of the CEBAF Element Database," ICALEPCS 2011, Grenoble, France, MOPKN029 http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/mopkn029.pdf