ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-TUPLE07

OVERVIEW OF FRIB'S DIAGNOSTICS CONTROLS SYSTEM*

B. S. Martins†, S. Cogan, M. Konrad, S. M. Lidia, D.O. Omitto, P. J. Rodriguez,
Facility for Rare Isotope Beams, Michigan State University, East Lansing, MI, USA

Abstract

In this work we will present an overview of the diagnostic systems put in place by FRIB's Beam Instrumentation and Measurements department. We will focus on the controls and integration aspects for different kinds of equipment, such as picoammeters and motor controllers, used to drive and readback the devices deployed on the beamline, such as profile monitors, Faraday cups, etc. In particular, we will discuss the controls software used in our deployment and how we make use of continuous integration and deployment systems to automate certain tasks and make the controls system in production more robust.

INTRODUCTION

The Beam Instrumentation and Measurements department is responsible for a myriad of devices used in FRIB's beamline for data collection and monitoring of operational parameters, such as Allison Scanners, Profile Monitors, Cameras, Beam Position Monitors, Beam Current Monitors, among many others. This variety of devices poses a challenge for the controls system, which must accommodate a considerable amount of different software pieces necessary to drive and read these devices.

In order to achieve this goal, FRIB builds on the EPICS [1] ecosystem and a number of commercial technologies for its controls system that will be described in this paper.

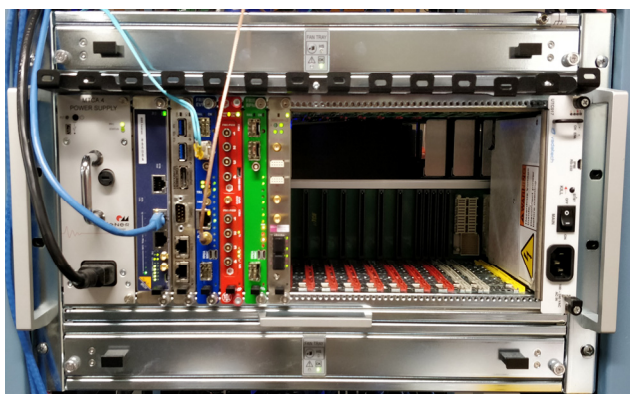

Figure 1: A µTCA crate with MCH, CPU, Event Receiver, Pico8, BPM and BCM cards.

DEVICES OVERVIEW

Diagnostics devices in the controls system can be categorized in a number of different ways. From the controls system infrastructure point of view, they can be divided into fast, medium and slow devices, according to the type and amount of data they generate per unit of time or how quickly they must signal for machine protection purposes. The category of a particular device guides where the device's EPICS IOC (Input/Output Controller) will be hosted, which can be one of the following: µTCA CPUs, Dedicated Server, Embedded or Virtual Machine.

µTCA CPUs

µTCA crates, as the one seen in Fig. 1, are typically used to host fast acquisition cards. We utilize a mix of commercial, off-the-shelf cards and in-house developed ones, with three acquisition card models in use:

- CAENels Pico8 picoammeter cards [2] for Faraday Cups, Profile Monitors, Neutron Detectors and Ion Chambers;
- FGPDB (FRIB General Purpose Digital Board, developed in-house) cards for Beam Position Monitors;
- Struck SIS8300 digitizer [3] for Beam Current Monitors.

In addition to any combination of the aforementioned acquisition cards, every µTCA crate also has one CPU card to host EPICS IOCs and one FGPDB card configured as an Event Receiver, used for timestamping acquisition data.

Dedicated Server Machines

There are a number of Gigabit Cameras in use at FRIB and each, if operated at full rate, can saturate a Gigabit Ethernet link, since frames are transferred without compression. Therefore, these cameras are considered fast acquisition devices. They are segregated into their own subnet and communicate exclusively with the respective EPICS IOC hosted on a dedicated server in order to alleviate network traffic for the rest of the controls system.

Embedded

Some of the Diagnostics devices have their IOC hosted by the device itself, either as a feature provided by their vendors or as a custom IOC developed by the Diagnostics group.

- Cryocon temperature monitors [4] for fast temperature measurements;
- Iseg modular power supplies [5];
- UEI Cubes [6] for standalone A/D and D/A converters.

While embedded IOCs are convenient use, they are harder to maintain since their software run on the device's custom hardware and depend on the vendor's ability and willingness to provide timely updates both for their own software and for the underlying operating system. In order to address this issue, there is a plan to develop new IOCs to control these devices from Virtual Machines, which can

* This material is based upon work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661, the State of Michigan and Michigan State University.
† martins@frib.msu.edu

ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-TUPLE07

be more readily kept updated without any input from device vendors.

Virtual Machines

Most of the Diagnostics IOCs are hosted by Virtual Machines, which can be easily resized, in terms of computing resources, to allow some flexibility. While there are a few VM-hosted IOCs that directly control physical devices, like Motor Controllers and Oscilloscopes, the majority of them are "soft" IOCs in the sense that they monitor the health of different systems, aggregate and process data collected from other IOCs, or coordinate the operation of different devices, which will be described in the next section.

MONITORING AND DATA PROCESSING IOCS

Apart from reading data, controlling and monitoring individual devices, an equally important role of the Diagnostics controls system is to coordinate the operation of different devices and present this information in a meaningful manner to operators and engineers. To that end, a number of "soft" IOCs were developed, which essentially consume data from certain EPICS PVs (Process Variables) for individual devices and expose combined or processed data on separate EPICS PVs. This way, any client interested in performing some relatively complex operation exposed by a "soft" IOC can simply make use of the exposed PVs, instead of having to implement the complex operation logic itself. The majority of the logic for these "soft" IOCs is implemented in Python, with the help of the pyDevSup module [7], which allows easy iteration during development and makes powerful Python libraries, such as numpy and scipy, available to the IOC.

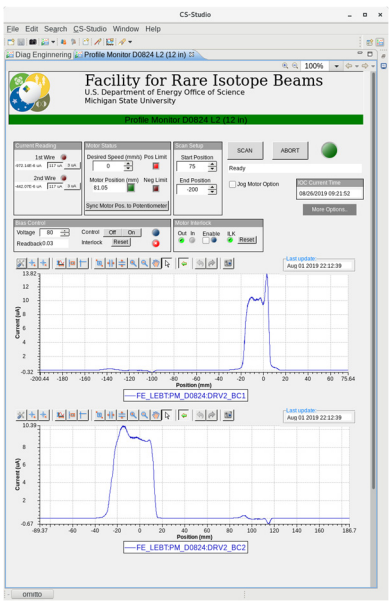

Figure 2: Screenshot of the Profile Monitor screen showing the beam profile captured by a scan.

Many "soft" IOCs are used at FRIB with varying degrees of complexity, but a good representative of this kind of IOC in the Diagnostics controls system is the one for the Profile Monitor, which interacts with several PVs from different IOCs in order to present a cohesive interface for beam profile scanning:

- Motor Controller PVs to drive the Profile Monitor wires in and out of the beamline;
- Safety PLC PVs to enable and disable the motor if the interlock status allows it;
- UEI Cube PVs to provide a bias signal to and read from a potentiometer that provides the Profile Monitor physical position along its travel;
- Pico8 PVs to read the electrical current through the Profile Monitor wires as they move through the beam.

By acting as this intermediate layer, the Profile Monitor IOC is able to provide a simple interface as EPICS PVs to the operators, with which they can easily specify the starting and ending position of a beam profile scan, run it, and obtain timestamped, correlated position and electric current data. The scans can be run by any client, be it GUI-based, like the one in Fig. 2, or scripted, with the sole requirement of being able to write and read EPICS PVs.

CONTINUOUS DEPLOYMENT / DELIVERY

All of the CPUs for Diagnostics IOCs are managed centrally and strive to have the latest Debian stable version as their operating system. Essential software, such as EPICS Base and several different modules, are packaged and released by FRIB's Continuous Deployment/Delivery [8] system as Debian packages. This system is comprised of a few off-the-shelf software tools:

- Jenkins [9]: automation server, used for building Debian packages and EPICS IOCs;
- Aptly [10]: Debian package repository management tool;
- Puppet [11]: infrastructure automation tool for deploying EPICS IOCs and Debian packages to the test and production machines.

The Continuous Deployment/Delivery system runs the following steps every time a change is detected in the repository of a FRIB Debian package:

1. Jenkins: fetch changes from the repository;
2. Jenkins: build the software;
3. Jenkins: run unit tests;
4. Jenkins: package binaries into Debian package;
5. Jenkins: test the resulting Debian packages;
6. Aptly: publish generated package to the packages repository;
7. Puppet: deploy new packages to the relevant machines.

There are two environments for which this entire process must run: the test and the production environments. If a package source code is to be changed, the changes must first land on the repository branch associated with the test environment. Once the changes are pushed, Jenkins will detect them and start the build process automatically. The

ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-TUPLE07

controls engineer responsible for the changes must wait until the build process is done and then manually run tests to check that the changes were successful on the test environment. If the engineer is satisfied with the tests, they must then merge the changes into the branch associated with the production environment, which will in turn start the process again, but this time the resulting packages will be made available on the production systems.

Diagnostics IOCs, while also being built and tested by Jenkins, do not get deployed as a Debian package. Instead, Puppet is used to fetch the source code and build IOCs on the target host when a change is detected in the source code of that IOC. The source code for IOCs also follow the two environments separation, test and production, each with an associated branch name in the repository.

The use of this two-tiered system increases confidence that the generated packages are sound, by having them tested in the test environment before deploying to production, and is robust and predictable since automation is used throughout the process.

Automated IOC building and testing

In addition to building Debian packages, Jenkins is also used for automatically building Diagnostics IOCs. Each IOC has an associated job that fetches the latest version of the IOC's code, builds it, runs it and collects all PV names from all specified instances. The collected PV names are then sent to a database, which allows Diagnostics controls engineers to have an overview of all the existing PV names in a central place.

STANDARD IOC MODULES

There are 28 different types of Diagnostics IOCs with 157 individual instances currently deployed in production. Each IOC type has its source code in its own version controlled repository. However, all Diagnostics IOCs are required to have the following common set of EPICS modules and services loaded and configured:

- recsync [12] for uploading all PV (Process Variable) names to a central, searchable location, the Channel Finder service;
- autosave [13] for preserving parameter setpoints across IOC restarts;
- caPutLog [14] for logging requested changes in PVs;
- iocStats [15] for monitoring IOC health;
- Channel Access Security configuration for partitioning write access to certain groups of PVs in order to prevent inadvertent and unauthorized access to operational parameters.

In order to facilitate and automate the inclusion and configuration of all these modules and services, a special EPICS module was developed for Diagnostics IOCs: *fribdiagstd*. This module is included in all Diagnostics IOCs and automatically links and configures the aforementioned modules and services. It also provides EPICS IOC templates for easily creating new IOCs with a standard configuration. For example, an IOC status overview page, shown in Fig. 3, shows all IOC instances deployed to the production environment. grouped by host, by displaying PVs from the iocStats module, which is one of the modules automatically included, loaded and configured by *fribdiagstd* in every Diagnostics IOC.

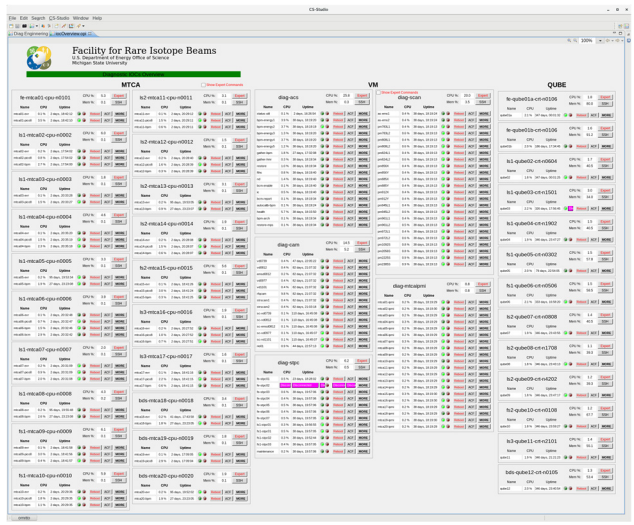

Figure 3: Screenshot of the Diagnostics IOCs overview page showing the status of all deployed IOCs.

CONCLUSION

The FRIB Diagnostics controls system is responsible for controlling a considerable amount of different devices, ranging from relatively slow motor controllers to fast data acquisition cards, and including "soft" IOCs written partly in Python that each combine and process PVs from cooperating devices into a set of related PVs.

Also, Diagnostics IOCs are hosted by in a few different kinds of hosts that have unique hardware and software characteristics. In order to tackle the resulting complexity of managing this many software pieces and kinds of hosts in a robust and reproducible fashion, off-the-shelf software tools are used as much as possible to automate the task of building, testing and deploying both fundamental supporting modules as Debian packages and EPICS IOCs themselves.

Moreover, standardization is achieved throughout all Diagnostics IOCs via the use of a common EPICS module developed in-house, *fribdiagstd*, which is included in every IOC.

REFERENCES

[1] EPICS, https://www.epics-controls.org

[2] CAENels AMC-Pico-8 picoammeter, https://www.caenels.com/products/amc-pico-8

[3] Struck SIS8300 digitizer, https://www.struck.de/sis8300.html

[4] Cryocon Model 18i, https://www.cryocon.com/ M18IProdFolder.php

[5] Iseg High Voltage MMS System, https://iseg-hv.com/en/products/systems#SYSTEM-MMS

ISBN: 978-3-95450-223-3 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2019-TUPLE07

[6] UEI Cube, https://www.ueidaq.com/products/3-slot-ethernet-based-i-o-data-acquisition-and-control-cube-with-powerpc-cpu-and-sd-slot

[7] pyDevSup, https://github.com/mdavidsaver/pyDevSup

[8] M. G. Konrad, D. G. Maxwell, and G. Shen, "Continuous Integration and Continuous Delivery at FRIB", in *Proc. PCaPAC'16*, Campinas, Brazil, Oct. 2016, pp. 145-147. doi:10.18429/JACoW-PCAPAC2016-FRITPLCO01

[9] Jenkins, https://jenkins.io

[10] Aptly, https://aptly.info

[11] Puppet, https://puppet.com

[12] recsync EPICS Record Synchronizer, https://github.com/ChannelFinder/recsync

[13] autosave, https://github.com/epics-modules/autosave

[14] caPutLog Channel Access Put Logger, https:// github.com/epics-modules/caPutLog

[15] iocStats EPICS IOC Status and Control, https://github.com/epics-modules/iocStats