ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

HIGH-LEVEL PHYSICS CONTROLS APPLICATIONS DEVELOPMENT FOR FRIB *

T. Zhang†, D. Maxwell, K. Fukushima, M. Ikegami and P. Ostroumov
Facility for Rare Isotope Beams, Michigan State University, East Lansing, USA

Abstract

For the accelerators driven by EPICS distributed control system, controls engineers solve the problem to make the devices work, while accelerator physicists dedicate themselves to make the machine run as the physics predicted. To fill the gap between the high-level physics controls and the low-level device controls, we developed a software framework so-called `phantasy` that can help the users like accelerator physicists and operators, to work well with the machine in an object-oriented way, based on which the implementations for the physics tuning algorithms could be very efficient, understandable and maintainable. Meanwhile, the modularized UI widgets are developed to standardize the high-level GUI applications development, to greatly reuse the codebase and ease the development. The most important thing is all the development also applies to other EPICS based accelerators. In this paper, the design and implementation for both interactive Python scripting controls and high-level GUIs development will be addressed.

INTRODUCTION

The driver LINAC of Facility for Rare Isotope Beams (FRIB) can accelerate all the stable isotopes to the kinetic energy higher than 200 MeV/u, deliver the energy of up to 400 kW on the end target, which will be more than two orders advancement in the heavy ion accelerators regime [1]. To achieve such goal, reliable and sophisticated applications for machine tuning should be ready for daily operation use.

Generally, high-level physics control is about controlling the accelerator with physics algorithms. The purpose is to apply the physics solution to the machine, and to expect the physics predicted machine behavior.

From the view of controls aspect, for the EPICS [2] driven facility, the entire machine is composed of many different kinds of devices, each of them is controlled by Input-Output-Controller (IOC) [3]. While all the IOCs are distributed in the same ether network, e.g. FRIB Controls Network, the data communication among these IOCs and any other client with the network access is defined by Channel Access (CA) protocol [4].

For client application implemented with a different programming language, specific software is required to be able to speak CA 'language'. For instance, `PyEPICS` [5] is the Python interface to CA.

On the other hand, accelerator physicists care more about the physics property and behavior, e.g. the magnetic field of a dipole, the gradient of a quadrupole, the simulated beam trajectory along with the accelerator, etc.

To bridge the device control and machine tuning on accelerator facility, systematic design of the software infrastructure for high-level physics controls is required.

The software framework should be able to help physicists establish a software environment for tuning algorithms development, here is the list of key problems to be resolved:

- Device control should be simple and easy enough to understand
- Implement physics algorithm should be convenient and maintainable
- Quick developing and testing should be supported

At FRIB, Python-based software solution for the high-level physics controls has been shaping gradually during the past few years, the project is named as `phantasy`, which stands for Physics High-level Applications aNd Toolkit for Accelerator SYstem [6]. Based on `phantasy`, various physics high-level applications are developed and deployed to FRIB Controls Network for the efficient beam commissioning. The next few sections go with the details about the development.

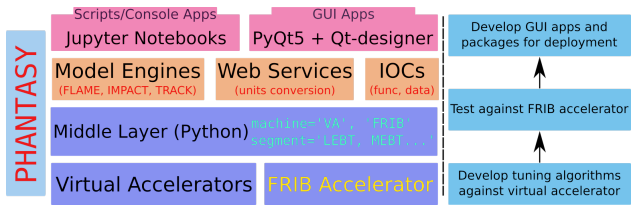

Figure 1: Development workflow of physics application with `phantasy`.

SOFTWARE FRAMEWORK OF `PHANTASY`

`phantasy` is designed to support quickly developing physics tuning algorithms on EPICS-based accelerator controls system. It is expected that accelerator physicists with Python knowledge could write scripts to control the accelerator by properly importing and using functionality provided by `phantasy`. Figure 1 shows the typical physics application development workflow. The virtual accelerator application which is part of the essential `phantasy` toolkit could be used with algorithm prototyping, once the algorithm is developed, the same script can test against FRIB accelerator

* 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.
† zhangt@frib.msu.edu

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

without any change, except the target machine and segment controlled by just one single line of code. Eventually, a dedicated GUI application will be created based on the proven tuning algorithm for the accelerator operation.

Device Abstraction

In the EPICS controls network, the device control is handled by the data communication with IOCs. The ultimate controls variable is the so-called process variable (PV), each PV is the control knob for device reading and writing, e.g. to monitor or change the stimulated current applied on the magnet device.

There are so many PVs distributed in the controls network, each PV name usually is a long string, though the naming convention is followed, it is still not simple to incorporate these long strings to construct a script.

From the object-oriented programming view, the device should be abstracted, such that the way of device control will be the communication between abstracted object rather than PVs. The most important is all the PVs could be managed separately, while the object for communication is always the one appears in the tuning script. In other word, the developed script should work with other segments of the same accelerator or even other accelerators.

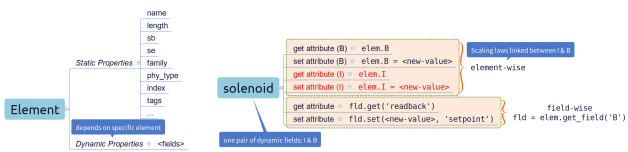

Figure 2: Device abstraction with `phantasy`.

In the framework of `phantasy`, the device abstraction is designed to integrate all the device-related information into one object, including the aforementioned PVs, which are termed as dynamic fields, and other properties attached to the device are termed as static fields. All these fields are equally abstracted into the Python object attributes, the user can get and set the attribute value by dotted-syntax. Here is an example (see Fig. 2), `elem` is an abstracted object, `I` is one of the attributes, then `elem.I` is to read the current `I` value, while `elem.I = 1.0` is to write a new value (here is 1.0) to `I`, the attribute `I` may link with the real PVs (one or many) to stimulated current on the hardware, e.g. solenoid, the regarding control logic is handled by `phantasy`.

From the view of high-level physics controls, all the values read from the device are measured with the engineering unit, e.g. current is read with Ampère, while for the physics modeling convenience, the corresponding physics unit interpretation is necessary for the smooth physics algorithm scripting. What `phantasy` does is to create the corresponding physics field for the device object, while both of them are linked with the same PV configuration, but different value representation, the relationship between them is maintained by a separate web application, so-called UNICORN, which is short for UNIt COnveRsioN [7]. For the example in Fig. 2, `B` is the physics attribute w.r.t. `I`. From `B`, one can directly control the solenoid in the language of the magnetic field in Tesla, while the `I` attribute will be synchronized by the scaling laws between `B` and `I`. Now the physicist can freely work with the device without any concerns.

In the framework of `phantasy`, all the device PV information and static properties are maintained by a dedicated package named as `phantasy-machines`, which contains all the physicist interested devices from LEBT to the downstream LINAC of FRIB, as the commissioning progresses, the data for new devices will be put into. Furthermore, only the `phantasy-machines` package is required to make the developed scripts still valid on other facilities, `phantasy` provides the relevant toolkit to help generate the required files.

Architecture

As Fig. 1 shows, the physics applications development should be able to deliver final GUI apps with tested tuning algorithms, meanwhile, development productivity and maintainability counts. Figure 3 shows the diagram of the whole `phantasy` project, including how to abstract the entire accelerator, how to develop GUI apps, how to organize the code, etc.

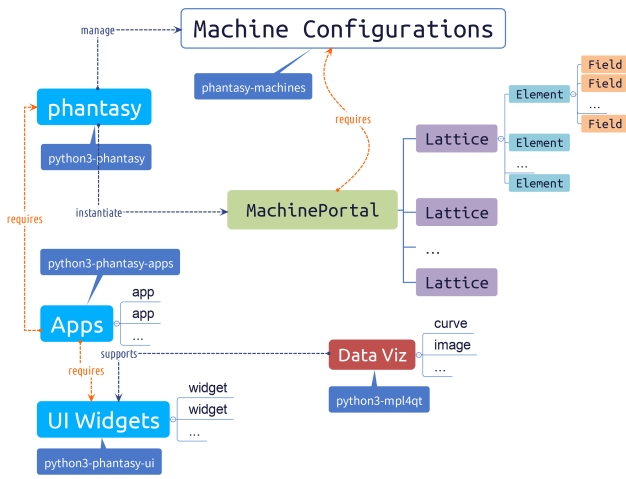

Figure 3: Overview of the architecture of `phantasy` project.

The class for machine abstraction is `MachinePortal`, which could be imported by `from phantasy import MachinePortal` in Python. Two arguments could be used for the instantiation of different machine and segment, e.g. `mp = MachinePortal(machine='FRIB', segment='LEBT')`. From `mp`, other lattices could be reached, e.g. `mp.load_lattice('LINAC')`. The loaded lattice is a sequence of elements (devices), each element is composed of multiple fields (see Fig. 2). All of the objects are self-explanatory, that is the user can know what does the object stands for by simply executing the object itself, thus, even the user with little knowledge of FRIB LINAC can still quickly master the machine and tune with physics algorithms with `phantasy`.

The users can use `phantasy` to interactively control the machine in any Python consoles, e.g. IPython or web-based

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

Jupyter [8, 9], etc., in Jupyter Notebook, the lattice object can even be represented in a much more user-friendly way, e.g. showing the sequence of devices in a well-organised table.

The GUI apps development is based on `phantasy`, highly modularized UI widgets are developed to standardize the app development and UI styles. Besides, a dedicated package so-called `mpl4qt` [10] is developed for the data visualization with `matplotlib` [11] in Qt5 GUI framework [12].

GUI APPLICATION DEVELOPMENT

There are two categories of the high-level physics GUI applications for FRIB, one is physics model-based and the other is non-model based, the former requires specific physics model configuration depends on the facility, the latter applies to other EPICS-based facilities. At FRIB, envelop tracking code FLAME [13] and particle tracking code TRACK [14] are used for beam studies. The model interface for FLAME has been implemented in `phantasy` framework. Additional work is required to interface other simulation code if physics online modeling is needed.

Below is the guideline the GUI apps development follows:

- Make the app self-explanatory as much as possible
- User-friendly and unified modern UI style matters
- Improve development efficiency by modularization
- Apps should be able to apply on any EPICS-based facility

UI Widgets for Qt5

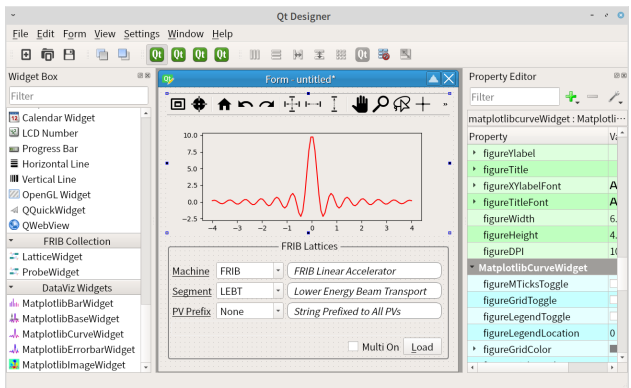

Figure 4: Qt-designer with developed UI widgets.

The GUI apps development for FRIB is based on the Python 3.x and Qt5, to maximize productivity and codebase reusability, Qt UI widgets have been developed. Figure 4 shows how to use `LatticeWidget` from `FRIB Collection` and `MatplotlibCurveWidget` from `DataViz Widgets` of Qt-designer to build the UI for a new Qt5 application. These UI widgets are developed as separate packages named as `phantasy-ui` and `mpl4qt`, respectively. All of these UI widgets are designed with simple usage, connecting the proper signals and slots can do whatever you want with the data from EPICS control system. Convenient command `makeBasePyQtApp` could be used to initiate the GUI app with unified project structure and UI styles, post-development could be continued.

Featured Online Apps

Figure 5 lists the high-level physics apps for FRIB commissioning, by using these apps, we significant boost the beam tuning efficiency [15]. The app launcher could be reached by right-clicking on the Desktop or File Explorer (Nautilus of GNOME) through `Physics Apps` context menu.

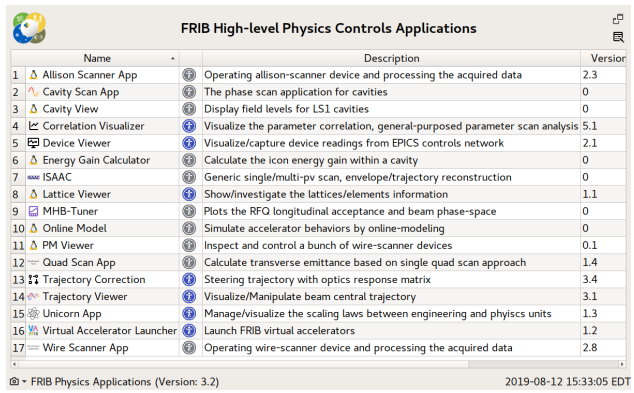

Figure 5: Gloabl launcher for FRIB physics apps.

Online Trajectory Correction The orbit-response-matrix (ORM) approach is applied to correct the beam central trajectory along the FRIB LINAC. Follow the development workflow shown in Fig. 1, the ORM-based trajectory tuning algorithm was developed and tested against virtual accelerator, then loading different lattices to work with FRIB LINAC, eventually, the GUI app was developed.

The definition of the response matrix item $\mathbf{R}_{i,j}$ could be defined by $\mathbf{R}_{i,j} := \frac{\Delta x_i}{\Delta \theta_j}$, where Δx_i is the trajectory change at i-th BPM, $\Delta \theta_j$ is the kick angle change of the j-th corrector. Usually, the kick strengths are controlled by varying the stimulated electric current applied on the correction magnets, thus the measured BPM response are against the kick strengths, not angles, but the responses are still valid for figuring out the optimal current settings for the correctors, so here we use θ to indicate the kick strength.

To facilitate the visualization of the trajectory and the selection of BPMs and correctors, `Trajectory Viewer` app is created (see Fig. 6). In the `Tools` menu, `Load Lattice` action could be invoked to change the lattice to work with, e.g. switching from `FRIB_VA` (development) to `FRIB` (commissioning).

After selecting BPMs and correctors in the `Monitors` and `Correctors` panels by checking/unchecking the checkboxes, one can launch another app for the ORM measurement and trajectory correction from `Tools` → `Trajectory Response Matrix`, or simply press keyboard shortcut: `CTRL + SHIFT + M`.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

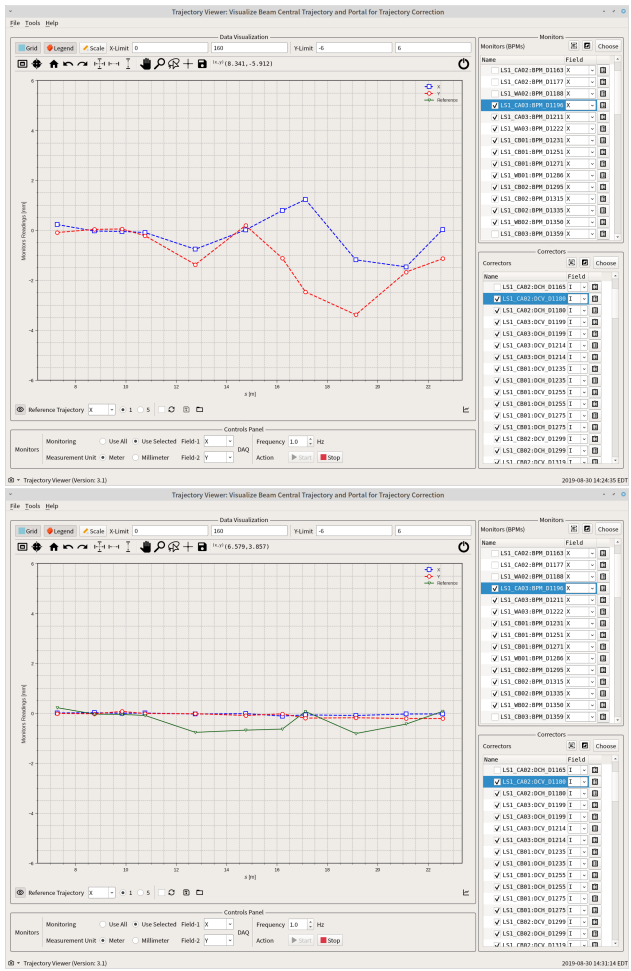

Figure 6: Trajectory Viewer with virtual accelerator BPM signal, upper: before correction, lower: after correction.

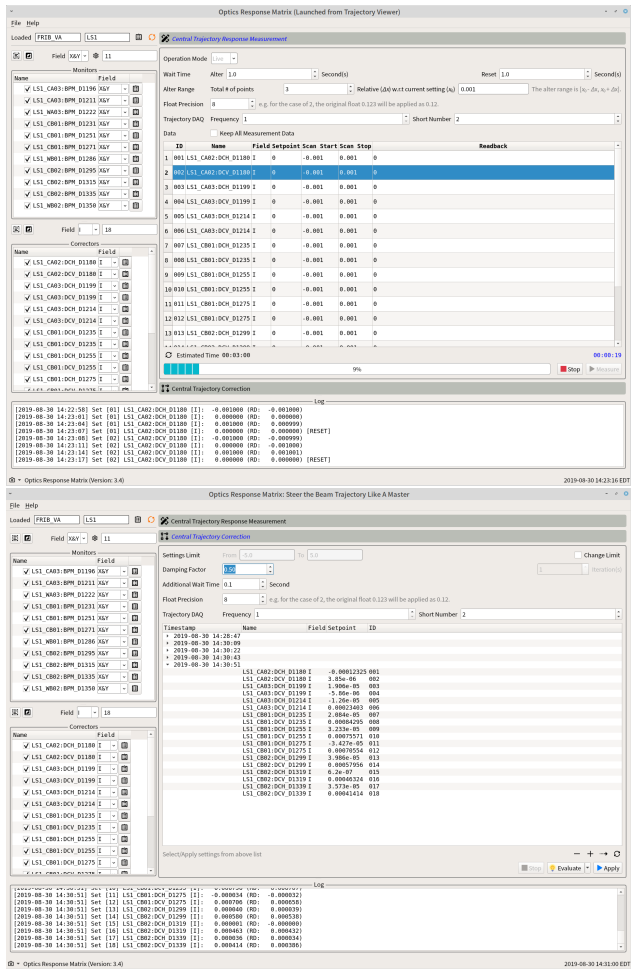

Figure 7: Correct trajectory by ORM approach, upper: ORM measurement, lower: correct trajectory with ORM.

The upper subfigure of Fig. 7 shows how the ORM could be measured. It is very convenient to change the sweeping range for each corrector, and also comes with featured buttons to control the measurement procedure. Once the measurement is done, menu actions are created to save/load the measured ORM to/from a file. The lower subfigure of Fig. 7 shows how the ORM can be used to correct the trajectory. All the calculated corrector settings should be confirmed by the user before applying, while all the corrector settings are stored as history marked with timestamps, which supports retrospection if any unexpected issue happens. By these two apps, we efficiently steered the beam central trajectory within ± 0.5 mm along the entire LS1 and part of FS1 section of FRIB LINAC [15].

Online Modeling After abstracting the accelerator with `phantasy`, one can get the physics model depends accelerator representation, usually, a lattice file that required by the specific simulation code to output the simulated machine physics behavior. Up to now, FLAME is supported to either generate a Python object for the physics model, or a lattice file for offline simulation. The common APIs for other physics model engines are under development.

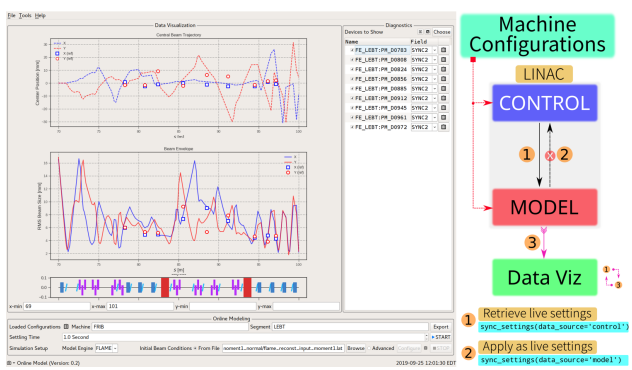

Figure 8: Online modeling for FRIB LINAC, left: online model app, right: schematic of the online-model dataflow.

Since FLAME can simulate the beam envelope and trajectory of FRIB driver LINAC within milli-second order [13], online modeling the machine at 1 Hz rate is achievable. Figure 8 shows the developed application for online modeling with FLAME at FRIB. Still, the common rule applies, that

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

is changing working lattice from the `Tools` menu to model the different machine or segment.

Generally, in the high-level physics controls implemented by `phantasy`, physics model (`MODEL`) and accelerator (`CONTROL`) working environment coexist in the memory of current working space. The dataflow between `CONTROL` and `MODEL` is controlled by the method of lattice object (`lat`), e.g. pull all the device live settings from `CONTROL` to `MODEL` is achieved by `lat.sync_settings(data_source='control')`, the device settings in physics unit will be automatically updated to model settings. At this point, one can output a model representation (e.g. a lattice file) for simulation study, or directly run the model. Once the simulation results are ready, just signal the dataviz widget with proper formatted data structure to the corresponding slot to update the plot. This is basically how the online model app works.

While on the other hand, the physics model could be well optimized based on various tuning ideas, which will finalize with one group of model settings, in the opposite direction (shown as ② in Fig. 8), one can push the model settings to the `CONTROL` working space by changing the `data_source` parameter to `model`, such workflow is model-based online optimization.

Concerning the discrepancy between the physics model and the real LINAC, the model-based online optimization has not been heavily applied on FRIB.

Parameter Correlation Analysis&Visualization By altering one device setting and monitoring the other devices responses, simple correlation may help the machine tuning, such procedure happens very often in the commissioning. One dedicated app has been built to make the parameter correlation study by just mouse clicking. Figure 9 shows the main user interface of `Correlation Visualizer` application, which is built upon `MatplotlibErrorbarWidget` of `mpl4qt` and `phantasy-ui`, as well as the underlying functionality from `phantasy` core.

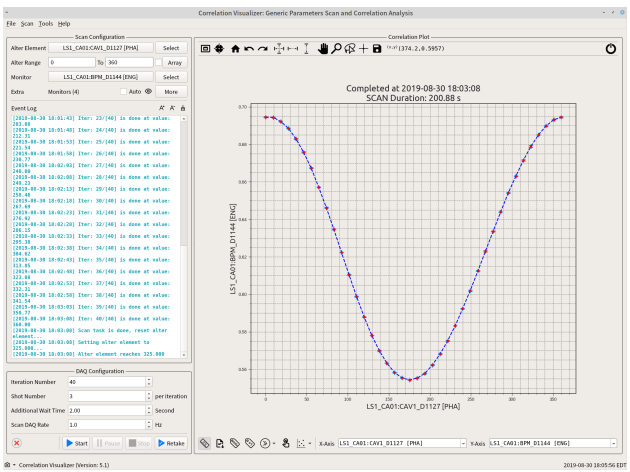

Figure 9: Correlation Visualizer app with data from RF cavity phase scan routine against virtual accelerator.

Highly abstracted device fields are listed in the `Select` popup dialog, one can select/deselect the dynamic field of any devices by checking/unchecking the item shown in Fig. 10, multiple selection is supported when selecting extra monitors. For the case of a device not been built into `phantasy-machine`, the full PV names should be put into the `Input PVs` section.

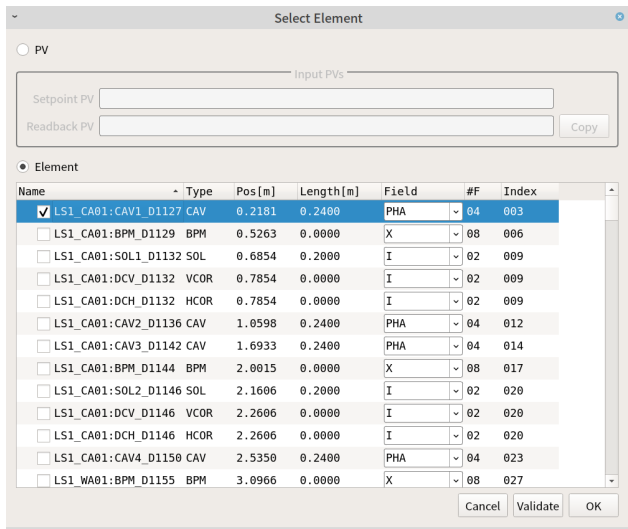

Figure 10: Object-oriented implementation of element selection of Correlation Visualizer.

The scan range and data acquisition configuration can be configured as the widgets and tooltips guided. Push `Start` button to start the scan, `Stop` to stop. If `Pause` is pushed, the current running job will be paused, push the `Resume` button (still is `Pause` button but different literal display name) to resume. The indicator on the left of `Start` button is the signal from Machine Protection System (MPS), if MPS indicator is not green, the running job will be paused, until the signal back to normal, and resumed by the user. The MPS integration could be bypassed by unchecking `Tools` → `MPS Guardian` menu.

The properties of the figure in the errorbar dataviz widget could be flexibly adjusted by `Config` menu action invoked by right-clicking context menu. `mpl4qt` is maintained to support more features to the dataviz part of all the physics apps, e.g. Allison scanner app utilizes `MatplotlibImageWidget` to facilitate the user-friendly data visualization [16]. The publish-ready high-quality figure could be produced directly from the app.

After the beam traverses through the Carbon stripper, multi-charge states will be generated. Figure 11 shows the detected beam current signal of different charge states when sweeping the magnetic dipole in FS1 section downstream of stripper.

`Correlation Visualizer` also supports two-dimensional scan, which could be launched by keyboard shortcut `CTRL + SHIFT + H`, the implementation is to create another outer loop of one variable, while the current scan task as the inner loop of another variable, such a

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

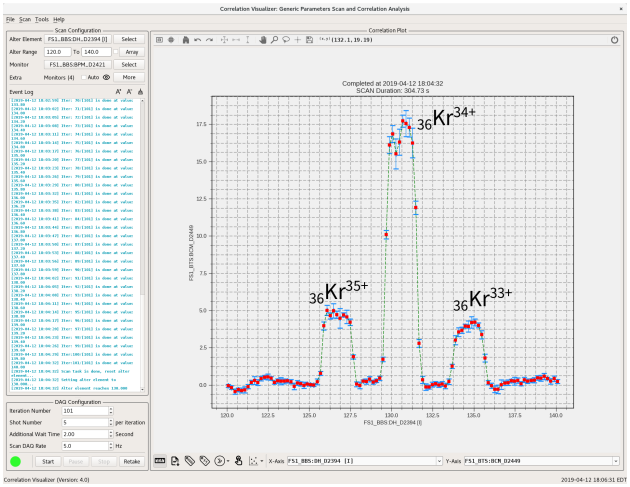

Figure 11: Beam current monitor signal v.s. magnetic dipole strength for the multi-charge states of Kr beam after Carbon stripper.

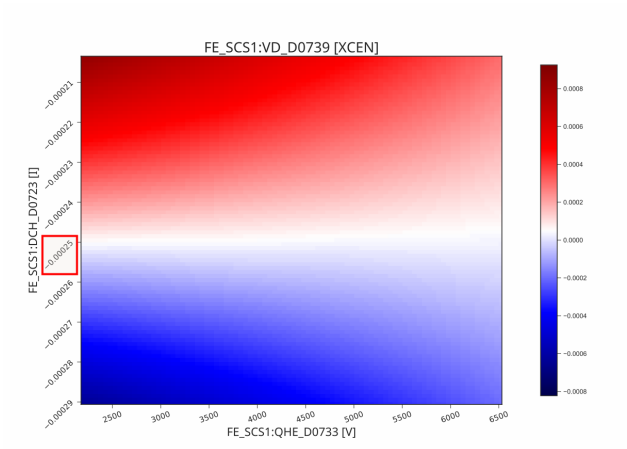

Figure 12: 2D parameter scan by Correlation Visualizer: beam-based alignment example, tested against virtual accelerator.

way can make the codebase reusable. Figure 12 shows the interpolated 3D density plot of the scan of one pair of corrector and quadrupole to do the beam-based alignment, the optimal setting of the corrector (red boxed) ensures the beam passes through the center of the quadrupole.

Software Management and Deployment

All the source code of physics applications are managed by git version control system [17], which allows us to track all the code changes and smoothes the collaboration. Currently, all the development is hosted in FRIB intranet, while `phantasy` project is starting to migrate to GitHub [18] for the public access.

Once the development reaches a new milestone, e.g. new features are developed, bugs are fixed, a new release will be tagged, after code review, the new release will be merged into production branch to create a new Debian package by the Jenkins service [19], then the package will be deployed to the FRIB Controls Network by Puppet service [20]. Such continues integration (CI) and continues delivery (CD) workflow has already been established and used every day at FRIB [21, 22]. As of writing this paper, all the physics apps are packaged for Debian Stretch OS, while as the IT infrastructure is evolving, all the development can be tuned up for the new updates, such workflow ensures the code quality, system reliability and efficiency.

For the physics applications development, we developed an automatic workflow to generate the VirtualBox appliance by Vagrant [23] for other users to test and develop the apps on their computer. And also developed another web-based computing platform for the app development, the users can use the web browser to do all the work regarding physics apps in the Jupyter Notebook in a private user space [24].

CONCLUSION

The systematic software solution based on Python programming language for high-level physics controls are designed and implemented at FRIB. The accelerator devices are abstracted into Python object with `phantasy`, controls in the interactive scripting environment is achieved, agile development workflow for the physics tuning algorithms are established. User-friendly GUI applications based on Qt5 are developed for the commissioning and operation. Generic Qt5 UI widgets are developed for building modularized applications. By using these apps, efficient machine tuning was achieved at FRIB LINAC, and in the same framework, by creating new machine configuration, we can continue to efficiently tune the downstream section in the upcoming milestone. All the physics apps are seamlessly integrated into FRIB's CI/CD system. The framework of `phantasy` also applies to other EPICS-based facilities.

ACKNOWLEDGMENTS

The authors would like to thank T. Yoshimoto, T. Maruta, A. Plastun, D. Chabot, S. Cogan, M. Konrad, B. Martins, D. Omitto and S. Lidia for the useful discussions.

REFERENCES

[1] J. Wei *et al.*, "Advances of the FRIB project", Int. J. Mod. Phys. E 28, 1930003 (2019). doi:10.1142/S0218301319300030

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

[3] EPICS IOC application developer's guide, https://epics.anl.gov/EpicsDocumentation/AppDevManuals/AppDevGuide/3.12BookFiles/AppDevGuide.book.html

[4] Channel access protocol specification, https://epics.anl.gov/base/R3-16/0-docs/CAproto/index.html

[5] Epics channel access for Python, https://cars9.uchicago.edu/software/python/pyepics3

[6] T. Zhang, "Physics high-level applications and toolkit for accelerator system", in *EPICS Collaboration Meeting*, Argonne National Laboratory, IL, USA, Jun. 2018, https://epics.anl.gov/meetings/2018-06/talks.html

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR07

[7] Web application for unit conversion, https://github.com/phantasy-project/unicorn-webapp

[8] IPython, https://ipython.org

[9] Jupyter, https://jupyter.org

[10] Qt widgets developed with matplotlib, https://phantasy-project.github.io/mpl4qt

[11] Matplotlib, https://matplotlib.org

[12] Qt GUI framework, https://www.qt.io

[13] Z. He *et al.*, "The fast linear accelerator modeling engine for FRIB online model service", Computer Physics Communications 234, 167 - 168 (2019). doi:10.1016/j.cpc.2018.07.013

[14] TRACK: the beam dynamics code, https://www.phy.anl.gov/atlas/TRACK

[15] P. Ostroumov *et al.*, "Beam commissioning in the first superconducting segment of the Facility for Rare Isotope Beams", *Phys. Rev. Accel. Beams.* 22, 080101 (2019). doi:10.1103/PhysRevAccelBeams.22.080101

[16] T. Zhang *et al.*, "High-level Application for the Emittance Measurement by Allison Scanner", presented at the North American Particle Accelerator Conf. (NAPAC'19), Lansing, MI, USA, Sep. 2019, paper TUPLS05. doi:10.18429/JACoW-NAPAC2019-TUPLS05

[17] Git, https://git-scm.com

[18] GitHub repository of 'phantasy' project, https://github.com/phantasy-project

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

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

[21] M. Konrad *et al.*, "Continuous Integration and Continuous Delivery at FRIB", in *Proc. 11th Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16)*, Campinas, Brazil, Oct. 2016, pp. 145–147. doi:10.18429/JACoW-PCAPAC2016-FRITPLCO01

[22] M. Konrad *et al.*, "Automatic deployment in a control system environment", presented at ICALEPCS'19, NY, USA, October 2019, paper MOPHA074, this conference.

[23] Vagrant, https://www.vagrantup.com

[24] T. Zhang *et al.*, "Cloud Computing Platform for High-Level Physics Applications Development", presented at ICALEPCS'19, NY, USA, October 2019, paper MOPHA167, this conference.