ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA135

ONLINE SIMULATION FRAMEWORK THROUGH HTTP SERVICES *

K.A. Brown†, M. Harvey, Y. Jing, P. Kankiya, S. Seletskiy,
Collider-Accelerator Department, BNL, Upton, NY, USA

Abstract

The development of HTTP service interfaces [1] to the BNL Collider-Accelerator Department (C-AD) controls system opens up the ability to more quickly and easily adapt existing codes developed for other systems for use at RHIC. A simple particle accelerator online model built for commissioning the NSLS II [2] was adapted for use with the Low Energy RHIC electron Cooling project (LEReC) [3] and the Coherent Electron Cooling (CeC) [4] proof of principle experiment. For this project, a set of Python modules and a Python application were adapted for use in RHIC by replacing NSLS II control system interfaces with Python modules that interface to the C-AD controls HTTP services [5]. This paper will discuss the new interfaces and the status of commissioning them for operations.

INTRODUCTION

Traditionally, sharing code from other facilities has been a labor intensive process. Control systems from different facilities have different communications interfaces, database interfaces, and even the general, non-control systems tools can greatly differ. To some level making control systems open source projects, as has been done with EPICS [6] and TANGO [7], makes this process easier. The C-AD controls system Accelerator Device Object (ADO) model [8, 9] is similar to TACO (the predecessor to TANGO) and over twenty years of refinement has become a robust, reliable, and very high performance system. Unfortunately, it is not open source and so it can be challenging to adopt code from other laboratories.

As much as these various control systems are different, they are, in general, very much the same, following what has long been known as the 'standard model' for accelerator controls [10, 11]. In general, for all of these control systems, the abstraction of a 'Device' is a fundamental structure. The details on how the parameters from a given Device are published to the rest of the control system are different, but in each system, if you know the name of a parameter, you can *get* or *set* the value and properties of that parameter.

The C-AD controls system has recently adopted a RESTful set of communication protocols that allow applications from many different platforms and operating systems to access controls Devices. These HTTP server interfaces also ease the task of adopting application level code from other facilities and control systems. In this report, we will discuss the process of adopting a Python application built for commissioning NSLS II for use in RHIC through the use of HTTP server interfaces.

* Work performed under Contract Number DE-SC0012704 with the auspices of the US Department of Energy.
† kbrown@bnl.gov

CONTROLS INTERFACE

The Device Server allows direct *set* and *get* communication to the ADO parameters while the Data Server provides access to logged data. For more detailed descriptions of the Controls interfaces see [1, 5]. The dataflow model for the simulation framework using the RESTful interface is shown in Fig. 1.

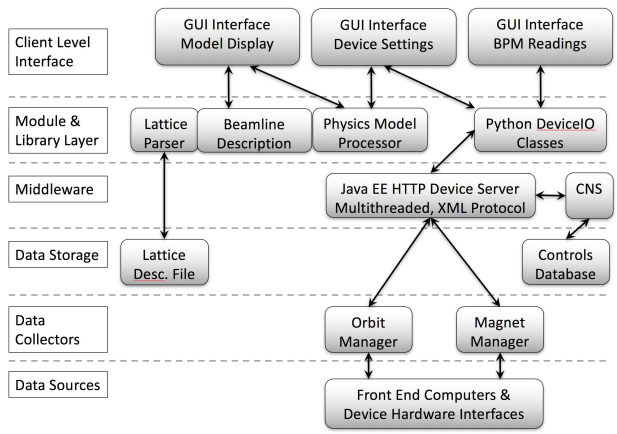

Figure 1: Data Flow and Interfaces for Simulation Framework.

For the online simulation framework, the names of the devices for a given model instance are listed in a model lattice description file, which has a syntax similar to standard accelerator physics model engines, such as madx or mad8 [12]. An example is shown in Fig. 9, in the appendix (ellipses indicate more similar elements follow.) The general format is *element_name: element_type, parameters*.

For this model lattice description to be connected to the control system, the *element_name* simply needs to be the ADO parameter name for that element.

LEREC & CEC

Both LEReC and CeC are electron accelerator systems, with a significant level of complexity. Each has electron guns with low energy sections and beam transport systems composed of standard accelerator components, including transport solenoids. Each is instrumented with current monitors, beam position monitors, and various types of profile monitors.

Figure 2 shows the layout of LEReC. The same electron beam is used to cool beams in each of the two RHIC rings. Since the ion beams in RHIC travel in opposite directions, the electron beam first passes by the ion beam in the RHIC Yellow ring and then is bent 180° and passes by the ion beam in the RHIC Blue ring, and then dumped. For the model

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA135

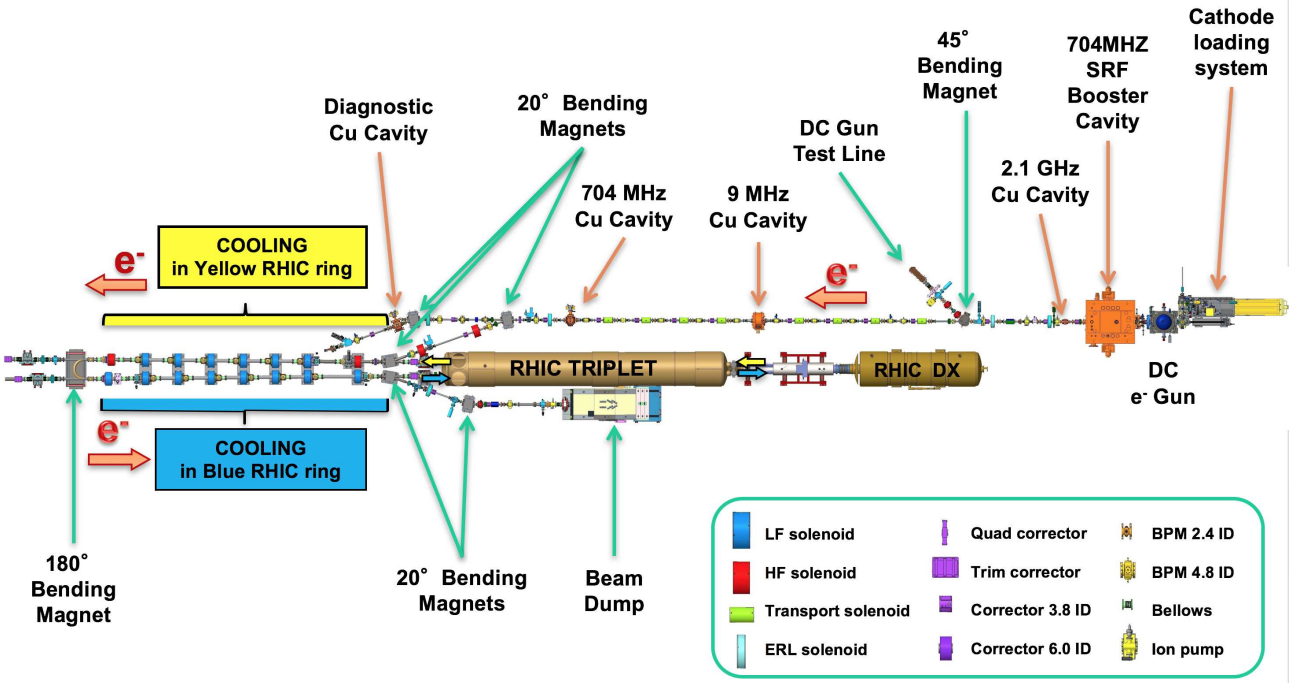

Figure 2: LEReC Layout (not to scale).

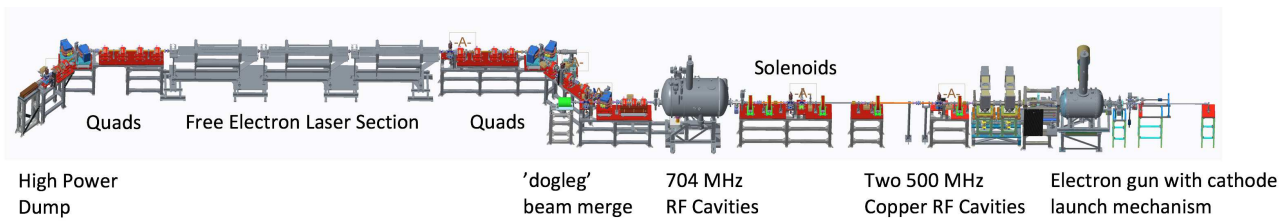

Figure 3: CeC Layout.

description, just the standard magnetic elements are used (drifts, bends, quads, solenoids, correctors, etc.), as well as the BPMs. Everything else is either a marker or treated as drift space.

Figure 3 shows the layout of CeC. Since this is an experiment to test the principle of coherent electron cooling, it is designed to only cool one bunch in one of the RHIC rings, at a low energy. The first section of quadrupoles, just after the 'dogleg' is where the electron beam images the ion bunch signal. The Free Electron Laser (FEL) amplifies the signal and in the second quadrupole section the electron beam applies a small kick to the ion bunch. For a more detailed description of the physics see [4]. What is important for this discussion is the model of the electron beams will make use of the standard magnetic optical elements, but the FEL will simply be seen as a drift. It is an extremely simple model and does not model the CeC process. As with LEReC it is meant just to describe basic optics conditions. More refined/precise models are used in offline analysis.

LEREC & CEC ONLINE MODELS

The simulation framework software is based on a high-level application used for NSLS II beam commissioning [2]. The idea behind this system is to use a fast, simple model to roughly describe the conditions of the accelerator, to allow the basic systems to be checked out and beam conditions to be initially established.

The general idea behind the software was to combine interactive simulations with measured beam positions into a single tool. The NSLS II application was built in Python, using models and interfaces developed at NSLS II. To convert the application over for use with LEReC involved disconnecting the NSLS II dependancies from the software. This meant replacing the controls interfaces used for NSLS II with our own. Figure 1 shows the data flow and interfaces for the new application.

For the new application, a new class was constructed, *apDeviceIo*, to use the C-AD Device Server, using a REST services protocol. Through this interface all controls data

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA135

is acquired. This class effectively replaces all the NSLS II controls calls, minimizing the changes needed to the original Python scripts. The names, types, and strengths of the accelerator devices are given in the lattice description file. This file holds all the beamline specific information. Except for this file, everything else is generic and highly modular. So, for CeC and LEReC, the only change to run the given simulation interface is to use a different lattice description file. A command line switch, at run time, allows the user to select what lattice description file to use.

```
import accphys3 as ap2
import ap_lattice_parser as clp3
import ap_auxiliary as ca
import apDeviceIo as dio
```

Figure 4: List of in-house modules built for the simulation framework.

There were three other modules, outside of the main script, that were imported for the NSLS II code. These modules, in principle, didn't need to change to adapt the code over the C-AD controls system, but did need to be modified to meet LEReC and CeC requirements.

In Fig. 4, the *accphys3* module contains the model engine components. The original implementation is basically simple linear matrix operations for first order optics. For LEReC and CeC, solenoids and skew quadrupoles needed to be added. Also, since the electron beam energies are relatively low, a simple space charge model was added. Except for minor modifications (improvements), *ap_lattice_parser* and *ap_auxiliary* remain the same as in the original code base.

GUI Displays

The main application consists of 3 separate GUI displays, a main menu window (Fig. 5), that allows selecting what beamline(s) to display, a spreadsheet like display containing some control buttons and the listing of the magnetic elements (Fig. 6), and a graphics display showing the model and measured orbits, a simple graphical description of the beamline elements, and a plot of the model beam sizes along the beamline (Fig. 7).

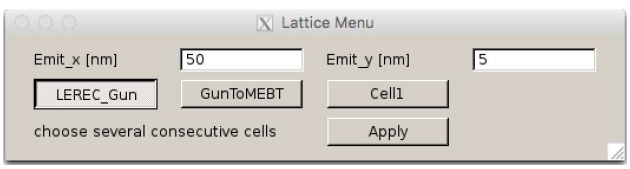

Figure 5: LEReC main menu page.

Lattice Parser

The lattice parser module reads a lattice description file and returns a Python dictionary containing the list of elements, a beamline dictionary if there are multiple beamlines

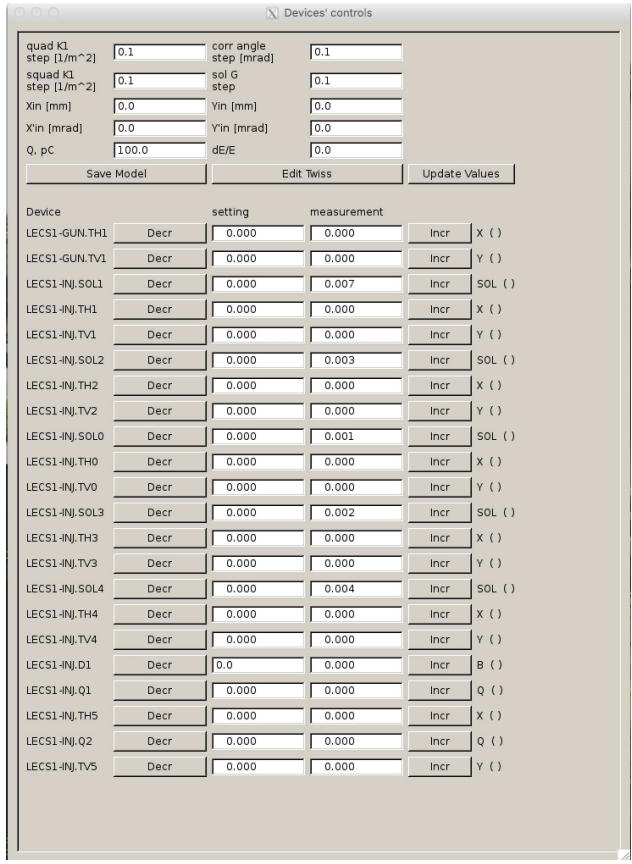

Figure 6: LEReC element listing page.

or if a single beamline is broken up into multiple lines, and a Twiss dictionary, containing the current set of optics parameters calculated by the model. When the parser runs it initializes all values based on the defaults given in the lattice description file. These values are temporary and are never used. Before bringing up the graphics display, the element strengths and Twiss values are recalculated based on live data.

Beamline Description

The *ap_auxiliary* module just structures the data for the graphics display of the beamline, or the 'Lattice' graph in the middle of the graphics display (see Fig. 7).

The application holds beamline and optics data in Python dictionary objects labeled consistently with the description from the lattice file. This is convenient for addressing data elements with labels.

Physics Model Processor

The *accphys3* module contains the matrix operations for calculating the optics parameters from the magnet strengths. These operations assume the input parameters are in physics units strengths (e.g., a bend angle in mrad). The controls system power supply ADOs work in engineering units (e.g., current in Amps). So either the simulation framework has to have transfer functions embedded in the Python code or another manager layer must exist to get the magnet strengths in

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA135

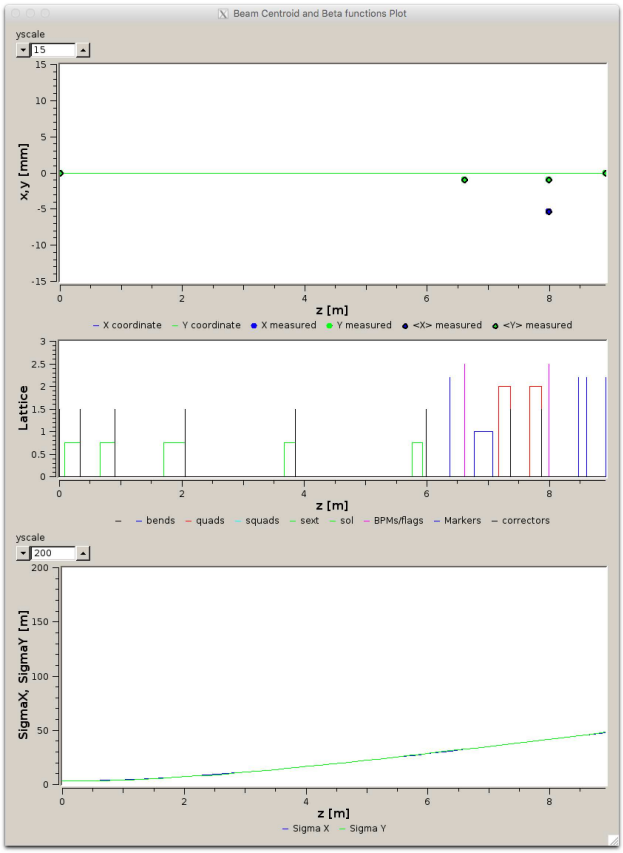

Figure 7: LEReC graphics page.

the correct units. For both CeC and LEReC another magnet manager layer exists as part of the controls system. This approach allows the strengths in physics units to be published and available to any application and be accessed using the HTTP device server interface.

MAGNET MANAGERS

The Magnet manager is a layer of abstraction between low level power supply controls and hig-level user interfaces which, for example, host the logic for the beamline model. The beamline model tools communicate with this layer to broadcast machine parameters such as beam energy. Power supplies that control magnet currents are represented in the controls systems as individual ADOs. The presence of the magnet manager allows representing magnets in the control system as ADOs and save magnet related properties in form of an ADO parameter such as K1 values in case of quad magnets. This also avails maintaining timed archives and data logs of magnet properties in physics units. At a change of energy in the beamline the magnet ADO converts physical units to power supply current based on a transfer function for each type of magnet. The transfer function is the relationship between beam energy, gamma, beam rigidity and magnet current coefficients. These transfer functions are provided by the system physicists. Magnet ADOs also store the reverse conversion from live current to corresponding rotation angles, bending angle, current gradient, and so forth, depending on the element. This scheme helps in determining the error between live and desired beam optics settings. The Magnet manager is an independent piece of software and has flexibility to take commands from any external tools. A generic Magnet manager class structure is shown in Fig. 8.

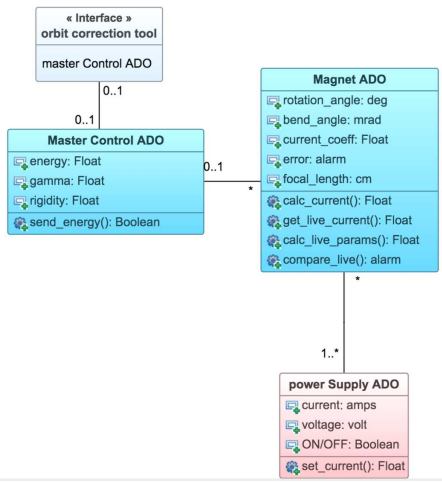

Figure 8: Magnet manager class structure.

SUMMARY

The process of borrowing code from another facility and built for another control system was very much simplified through the use of REST services interfaces. The software for the simulation framework is easily adaptable to multiple applications, just by creating and referencing a new lattice description file. As of the time of the writing of this paper, the software has been demonstrated to work correctly but has not been used for general operations. Both CeC and LEReC will have significant beam operations next year, and the online simulation environment will be tested and integrated into normal operations.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA135

APPENDIX

```
! listing of beam line elements
!
! the sol elements are solenoids
lecs1-gun.sol1: SOL, L=0.06, G=0.
. . .
! correctors
lecs1-gun.th1: hKick, L=0., kick=0.
. . .
! Quads
lecs1-inj.tq1: QUAD, L=0.05, K1= 0.
. . .
! Skew quads
lecs1-inj.tqs1: SQUAD, L=0.05, K1=0.
. . .
! Bend, 45 degree dipole
lecs1-inj.d1: CSBEND,  L=0.3089, &
          ANGLE=-0.7853981634, &
          E1=0.0, E2=0.0
! drifts
Cath_GCSol: DRIF, L= 0.12
. . .
! Markers
Cathode_Face: Mark
. . .
LECS1_End:     Mark
! monitors  - these are bpm's
lecs1-inj.b0:          Moni
. . .
!===== Twiss parameters ===
TWISS_0disp :  &
        BETA0, BETX = 9.35591, &
        ALFX = 0.663442, &
        BETY = 8.63852, &
        ALFY = 0.936493, &
        DX = 0.148315, DPX = 0.0
!====================
LEREC: LINE=(lecs1-gun.th1, &
               lecs1-gun.tv1, &
               . . .
               LECS1_End )
USE: LEREC
```

Figure 9: Example lattice description for simulation framework.

REFERENCES

[1] T. D'Ottavio *et al.*, presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUPHA157, this conference.

[2] S. Seletskiy *et al.*, in *Proc. IPAC'15*, Richmond, VA, USA, May 2015, pp. 1953-1955.

[3] A. Fedotov *et al.*, in *Proc. NAPAC'16*, Chicago, IL, USA, Oct. 2016, pp. 867-869.

[4] V.N. Litvinenko *et al.*, in *Proc. IPAC'11*, San Sebastian, Spain, Sep. 2011, pp. 3442-3444.

[5] K. Brown *et al.*, presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUPHA153, this conference.

[6] EPICS, http://www.aps.anl.gov/epics/.

[7] TANGO, http://www.tango-controls.org/.

[8] J. Skelly and J. Morris, in *Proc. ICALEPCS'99*, Trieste, Italy, Oct. 1999, pp. 42-24.

[9] L. Hoff and J. Skelly, in *Proc. ICALEPCS'93*, Berlin, Germany, Oct. 1993, Nucl. Instr. and Meth. A, p. 185, 1993.

[10] B. Kuiper, in *Proc. ICALEPCS 1991*, p.602, KEK, Tsukuba, JAPAN

[11] V.N. Alferov *et al.*, in *Proc. ICALEPCS 1991*, p.134, KEK, Tsukuba, JAPAN

[12] MAD, http://mad.web.cern.ch/mad/.