ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB100

A GENERIC SOFTWARE PLATFORM FOR RAPID PROTOTYPING OF ONLINE CONTROL ALGORITHMS

C. J. R. Duncan†, M. B. Andorf, V. Khachatryan, C. Gulliford,
J. M. Maxson, D. L. Rubin, I. V. Bazarov, Cornell University, Ithaca, NY, USA

Abstract

Algorithmic control of accelerators is an active area of research that promises significant improvements in machine performance. To facilitate rapid algorithm prototyping, we have developed a generic interface between accelerator controls, beam physics modelling software and modern scripting languages. The work-flow of a project using this interface begins with testing algorithms of choice offline in simulation. After off-line testing, the same code can be deployed on real machines via the Experimental Physics and Industrial Control System (EPICS) API. We include noise in our simulations in order to mimic realistic accelerator behaviour and to evaluate robustness of algorithms to experimental uncertainties and long-term drifts. The results of test cases of using this framework are presented, including emittance tuning of the Cornell Electron Storage Ring (CESR), correction of diurnal drift in CESR steering and orbit correction on CESR and the Cornell-BNL ERL Test Accelerator (CBETA).

INTRODUCTION

Particle accelerators are a natural application domain for software automation. But the lack of a community standard interface between algorithm code, machine controls and simulation software impedes collaboration between accelerator scientists at different facilities, as well as collaboration between accelerator and computer scientists. These frustrations can be eliminated by introducing in software an *abstraction layer* that presents a consistent interface to control code, consistent across different machines and consistent across machines and simulation software. In this proceeding, we describe two accelerator facilities at Cornell that make use of two different control systems. We then sketch the software components of a prototype abstraction layer. We report proof-of-concept results showing how, through our abstraction layer, (i) control code written for different accelerators can be shared, and (ii) third-party optimization algorithms can be rapidly tested in simulation. We end by describing a method for simulating noisy drift observed in CESR orbit data.

ACCELERATOR FACILITIES

The Cornell Electron Storage Ring (CESR) is the source for the Cornell High Energy Synchroton Source (CHESS) x-ray user facility. CESR comprises hundreds of independently powered electromagnetic elements and thousands of sensors that are controlled and monitored by a centralized Multi-Port Memory device (MPM). The components of the MPM system are schematized in Fig. 2.

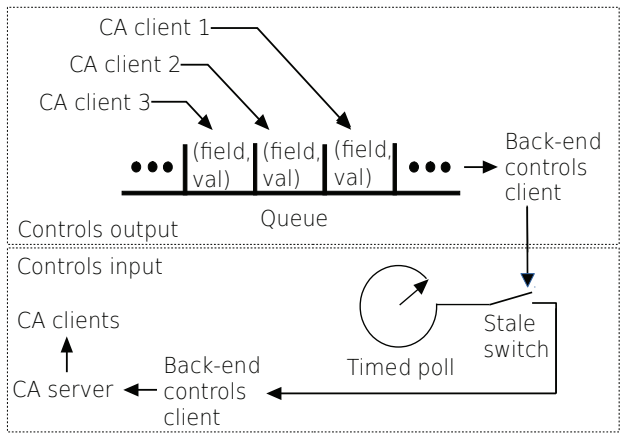

Figure 1: Schematic of interface software components. On the output side, EPICS Channel Access clients put commands on a work-queue that is managed by the thin software input-output controller (IOC). The IOC consumes the queue by translating and then passing commands to the back-end control and simulation software, which run as a separate processes. On the input side, a timer periodically polls the back-end for data, which triggers execution of simulation code. The simulation trigger is vetoed by a register ("stale switch") if and only if the simulation parameters are unchanged.

The Cornell-Brookhaven ERL Test Accelerator (CBETA) is a superconducting RF multi-turn energy recovery linac. Control and monitoring is handled by a distributed network of dozens of EPICS Input-Output Controllers (IOCs).

SOFTWARE COMPONENTS

Our abstraction layer interfaces with user code via the Channel Access (CA) network protocol, a component of EPICS. Client CA libraries exist for Python (*pyepics*), and Matlab (*labCA*). In our implementation, an EPICS IOC runs a CA server that handles the details of communicating client commands to control system and simulation software, as shown in Fig. 1. Since EPICS is designed to perform the same functions as the legacy control software, the details of the implementation consist in translating equivalent functionality, together with data book-keeping. Client commands to change command values ("puts") are placed by the IOC on a task-queue. Items on the queue are consumed by a worker thread that is connected to the legacy control system as a

† cjd257@cornell.edu

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB100

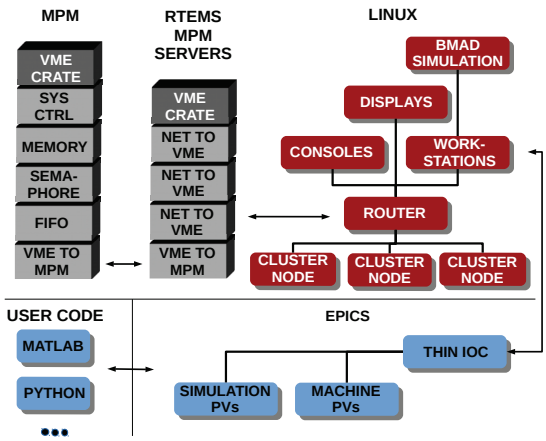

Figure 2: Block diagram showing how our EPICS interface integrates with legacy CESR control and simulation software: clockwise from bottom, user code interacts with both machine and simulation by setting and getting Process Variables (PVs) through the network Channel Access protocol; a thin Input-Output Controller running on a Linux workstation acts as the client of a VMEbus interface to the Multi-Port Memory (MPM) that aggregates machine command and sensor data.

client, using a look-up table to execute the equivalent legacy command.

CESR and CBETA beamlines are simulated using two codebases: (a) the *BMAD* library in compiled code and interactively in the command-line interface *Tao* [1]; (b) for simulating space-charge effects, the General Particle Tracer (GPT). In our implementation, an EPICS IOC hides simulation software from user code. Simulation parameters are mapped to IOC device inputs and simulation results to IOC device outputs. By default the IOC schedules execution of simulation code automatically. For automatic execution to be responsive to user input, the IOC must infer user intention. For example, a user might put new values to 100 steering magnets consecutively and then attempt to get the resulting orbit. The server infers that the sequence input of commands has terminated at the 100th value by introducing latency into the scheduling of the simulation, as follows. When the IOC receives a put, the IOC restarts a count-down timer, executing simulation code once the timer expires. In our example, because successive puts restart the count-down, the timer only expires after the 100th put is made. The timer duration is set to the time-cost of one simulation run.

CONTROL ALGORITHMS

Orbit Correction

Our aim in this section is to demonstrate our proposed work-flow, showing that a script tested in simulation via our interface can then be deployed on the real machine. The first example algorithm sets the strength of corrector dipoles to center the particle orbit on a reference trajectory. Suppose we have beam position monitors (bpms) at n positions in the beam-line and let x_i be the n corresponding observed horizontal (vertical) coordinates of the beam. Suppose further we have m horizontal (vertical) correctors and let θ_i be the corrector strengths in radians. We aim to center the beam at coordinates $\tilde{x}_i$ and thus seek a change to the correctors $\Delta\theta_i$ that solves the linear system,

$$x_i + \sum_j R_{ij}\Delta\theta_j = \tilde{x}_i; \quad R_{ij} := \frac{\partial x_i}{\partial \theta_j} \tag{1}$$

The matrix R admits a singular value decomposition $R = USV^\dagger$ and a solution to (1) is,

$$\Delta\theta_i = \sum_{j=1}^{\min\{n,m\}} \sum_{k=1}^{n} V_{ij} S_{jj}^{-1} U_{jk}^\dagger (\tilde{x}_k - x_k) \tag{2}$$

Using our interface, we implemented the algorithm for CESR in Python and for CBETA in Matlab, with results shown in Fig. 3.

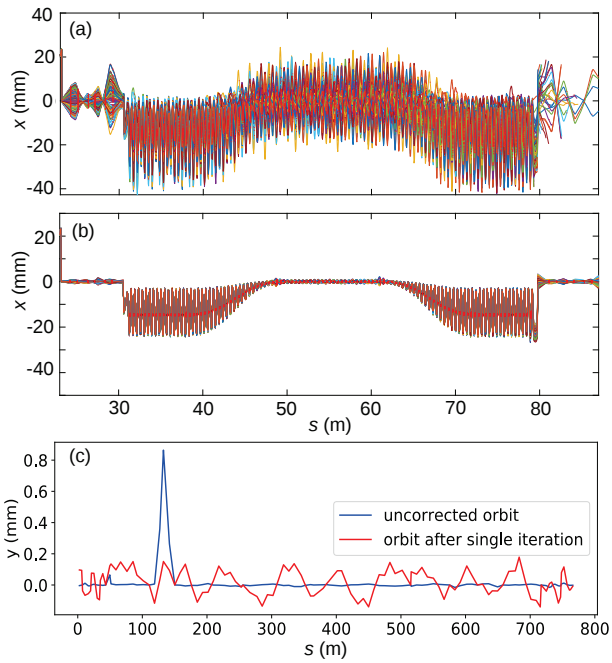

Figure 3: (a) Matlab script interfaces with *Tao* simulation of a CBETA lattice perturbed from design parameters, demonstrating the orbit correction algorithm Eq. (2). Ten traces per panel show different beam energies making one turn of the ERL racetrack uncorrected lattice; (b) converged result of applying the design response matrix iteratively; (c) Python script interfaces with CESR controls to demonstrate Eq. 2. Data are taken from bpm measurements, the response matrix is obtained from BMAD simulation.

Multi-Objective Optimization

A multi-objective optimization algorithm finds the set of optimal trade-offs between competing scalar valued objective functions (*trade-off frontier*). Such trade-offs arise correcting vertical orbit errors in CESR because correction

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB100

magnets introduce unwanted vertical dispersion. Magnet strengths represent an optimal trade-off if any further reduction in orbit error necessarily leads to a growth in dispersion or vice-versa. Genetic algorithms are suited to finding trade-off frontiers. *Platypus* is an open-source Python genetic algorithm package [2]. We give Platypus an objective function using our abstraction layer, letting this third-party software drive a BMAD simulation with the results shown in Fig. 4.

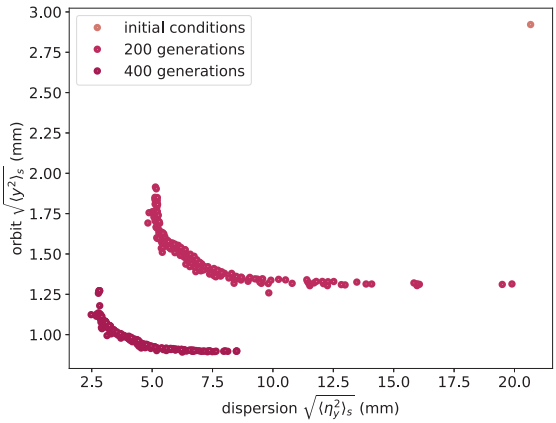

Figure 4: Trade-off frontiers in the space of orbit and dispersion for a misaligned lattice, as found by the Platypus genetic algorithm package driving a BMAD simulation via our interface.

SIMULATING NOISE AND DRIFT

Algorithmic control must reckon with both measurement uncertainties and machine drifts. It is therefore desirable to incorporate both these effects in simulation. Time series data collected at CESR shows random oscillations in kicker strength on a timescale of several minutes. We fit a multivariate Ornstein-Ulenhbeck process to this data, a model defined by the stochastic difference equation [3]:

$$x_i(t_{k+1}) = x_i(t_k) - \lambda_{ij}\Delta x_j \Delta t + \sigma_{ij}\sqrt{\Delta t}\xi_k. \qquad (3)$$

Here, σ_{ij} is a covariance matrix and the ξ_k are i.i.d. Gaussian random variables with mean zero and unit variance. A comparison of the measured and simulated time series is shown in Fig. 5. Kicker strengths are inferred from orbit measurements by reduced-chi-squared fits of an analytic, linear model, in which the fit parameters are the number of kickers n, the kicker locations (s, i) and kicker strengths θ_i:

$$x(s) = \sum_i^n \theta_i \frac{\sqrt{\beta_s \beta_{s,i}}}{2\sin(\pi q)} \cos(\pi q - \phi(x) + \phi_i), \qquad (4)$$

Simulating noise makes possible a self-consistency test of this inference procedure, with results shown in Fig. 5.

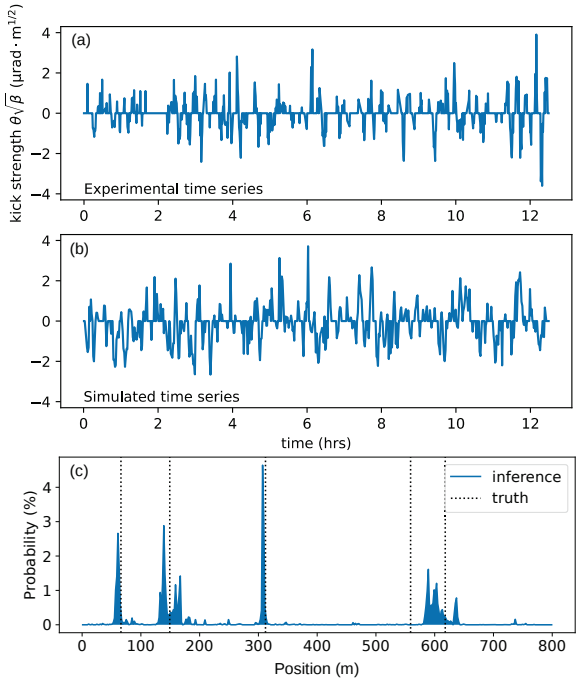

Figure 5: (a) typical kicker strength time series obtained from CESR orbit data; (b) kicker strength fluctuations simulated as an Ornstein-Ulenhbeck process, Eq. (3); (c) histogram of drifting kicker locations as inferred from fits to simulated data.

CONCLUSION

The value of the larger accelerator community developing a standard semantics for accelerator control code is clear. We have shown how such semantics could easily be implemented through an abstraction layer that leaves in place existing facility-specific control systems.

ACKNOWLEDGEMENTS

This work was upported by the US Department of Energy through grant number DE-SC 0013571. C.J.R.D., M.B.A., J.M.M. and I.V.B. partially supported by the US National Science Foundation under Award OIA-1549132, the Center for Bright Beams.

REFERENCES

[1] D. Sagan, "Bmad: A relativistic charged particle simulation library", *Nucl. Instr. And Meth.* A 558 (2006) 356

[2] Platypus, https://github.com/Project-Platypus/Platypus

[3] R. Sing, D. Ghosh, R. Adhikari, "Fast Bayesian inference of the multvariate Ornstein-Uhlenbeck process", *PRE 98* (2018) 012136.