ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL04

A MODEL-BASED SIMULATOR FOR THE LCLS ACCELERATOR

M. Gibbs*, W. Colocho, J. Shtalenkova, A. Osman,
SLAC National Accelerator Laboratory, Menlo Park, USA

Abstract

The Linac Coherent Light Source (LCLS) at the SLAC National Accelerator Laboratory is currently undergoing a major upgrade. In order to facilitate the development of new software that will be needed to operate the upgraded machine, a simulator of the LCLS electron beam, and accelerator devices that measure and manipulate the beam, has been developed. The simulator is comprised of several small "services" that simulate different types of devices, and provide an EPICS interface identical to the real control system. All of the services communicate with a central beam line model to change accelerator parameters and retrieve information about the simulated beam.

MOTIVATION

SLAC has spent the majority of 2019 with the LCLS accelerator offline while a new superconducting linear accelerator is installed upstream of the existing normal-conducting linear accelerator, and the original transport line from the linac to the hard x-ray undulator line is replaced with two new transport lines feeding two new undulator lines. This will fundamentally change the structure of LCLS from a "straight line" machine with one electron source and one beam destination to a machine with two electron sources, each of which can feed either a hard x-ray or soft x-ray undulator line. A large number of high-level software applications used for the operation of the accelerator (including applications to measure electron beam characteristics, automation tools to perform routine procedures, and tools to align and calibrate beam-line devices) need modification to support multiple beam-lines, or new hardware. The hardware installation schedule dictates that most of the software modifications need to happen before hardware is installed or connected to the EPICS control system.

An accelerator simulation system called Simulacrum was created to give software developers a way to test their applications against a simulated electron beam using an EPICS interface identical to what the real accelerator provides.

ARCHITECTURE

Simulacrum is a modular system, comprised of many independent device simulation services communicating with a single accelerator model service (see Fig 1). Each device service hosts the complement of EPICS process variables (PVs) that applications use to interact with the device type. When applications read or write to these PVs, the device service can inform the model of changes to device parameters, trigger a re-calculation of the simulated beam, and query the model for the state of the beam at a particular location.

* mgibbs@slac.stanford.edu

IMPLEMENTATION

All services are written in Python 3, and use the PyZMQ module [1] to communicate with the model service via ZeroMQ [2] messages over TCP. The services are typically all run from one computer, but the TCP-based communication provides a path to running the simulator across several computers, if the need arises.

Model Service

The model service is the only service that users *must* run. It hosts an instance of Tao, an accelerator modelling application that is part of the BMAD software toolkit for charged particle and x-ray simulations [3]. The model service takes a BMAD accelerator lattice definition as input, and instantiates an accelerator model. Because the model service is in charge of keeping the lattice definition, it is easy to simulate different accelerators, as long as their devices share the same EPICS interface. This is routinely used to switch between simulating the machine as it existed in 2018 to simulating the machine as it will exist when operations resume.

A request-response pattern is used to interact with the model: the model service sends its Tao instance a command, and a result is returned. This is the primary way other services interact with the model: they send a Tao command via ZeroMQ, the model service receives the command, runs it, and replies with the result of the command. It is also possible to open an interactive terminal to send and receive Tao commands - this is useful for debugging, and also allows expert users to configure parameters of the model that are unavailable via EPICS, like misalignment of devices and initial beam conditions.

In addition to the request-response communication, the model service has a second communication channel that operates using a publish-subscribe pattern. This channel is used to broadcast frequently-updating beam parameters (like trajectory) at 10 Hz. Any service can subscribe to these broadcasts, and update PVs (like beam position monitor signals) in real time.

Finally, the model service uses the p4p module [4] to act as a PVAccess server. It hosts a large NTTable PV that contains the beam's Twiss parameters at every element, along with the length of the element, the element's position along the beam-line, and the 6x6 transfer matrix for each element. This table is refreshed and re-published at 1 Hz.

Device Services

The device services are the clients of the model service, sending Tao commands and receiving information about the current state of the simulated devices or beam. When a device service starts, it queries the model service for a list of all the devices the service will simulate (for example, a list

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL04

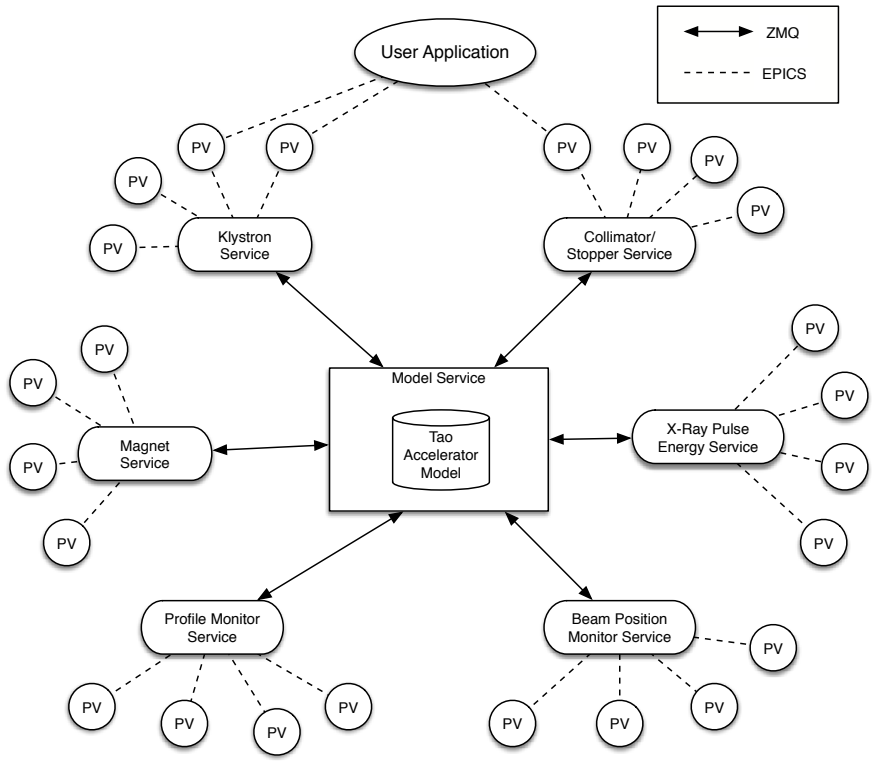

Figure 1: Simulacrum is structured as many independent processes communicating with a beam physics model.

of all quadrupole magnets), as well as the initial state of each of those devices. It then creates EPICS PVs for each device, setting the values using the initial state from the model.

The device services use the caproto [5] library to host EPICS PVs. The services define every PV needed to simulate one instance of the device type. Each PV can have getter and setter methods attached that are run any time an EPICS client reads or writes to the PV. These getters and setters often send messages to the model service to fetch data, or update the model parameters as device settings change. In this way, the device services can be thought of as a translation layer that takes inputs in the form of EPICS operations and translates them into a Tao command that the model uses, then translates the results of the Tao command back into changes in the EPICS PVs.

Another responsibility of a device service is to simulate details of the hardware that are not considered by Tao. For instance, the service that simulates LCLS klystrons implements a complex system of about 70 interlocks, and hosts the PVs that are used to diagnose and reset interlock faults.

Device services have been written to simulate seven device types: beam position monitors, insertable profile monitor screens, klystrons, magnets (including quadrupoles, corrector dipoles, and bend dipoles), adjustable collimators, beam stoppers, and a x-ray pulse energy measurement device. Additionally, a "generic PV service" was built to host static EPICS PVs. This service is mainly used to mock soft IOCs at SLAC that store things like the results of emittance measurements in PVs. The generic PV service is also a convenient place to host very simple, static versions of device data for device types which have not yet been implemented as services.

USE IN SOFTWARE DEVELOPMENT

One notable example of Simulacrum's use in software development has been the study of new features for Ocelot [6], a platform for automated optimization of accelerator performance. Ocelot allows users to define an objective function that maps accelerator device settings to a target output. The optimizer uses numerical algorithms to search this multidimensional parameter space defined by the objective function for optima by iteratively changing device parameters and sampling the objective function. At LCLS, Ocelot is used to optimize FEL pulse intensity by manipulating quadrupole magnets. Simulacrum was used to simulate interactions with the LCLS controls system during development of the Gaussian process optimization method over the accelerator downtime. Simulacrum was also used to develop the Ocelot interface for the SPEAR3 accelerator control system; at SPEAR3 Ocelot was used to minimize Touschek lifetime using the skew quadrupole magnet complement.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL04

A second example involves beam-based alignment (BBA) of LCLS undulator segments. A complex procedure executed by a software application is used to straighten the electron beam trajectory as it travels through the FEL undulator [7]. To verify that the application properly implements the BBA procedure, known misalignments of undulator segments can be injected into the accelerator model. The BBA application is then run against the simulator, and the calculated corrections are compared against the injected misalignments. If the two match, it is a strong indication that the software is working correctly.

FUTURE USES

Simulacrum is also useful as a training tool. Accelerator operators undergo exhaustive training in order to understand and interact with the myriad of systems and components necessary to operate the accelerator safely and effectively. Operators can use completely unmodified accelerator software to interact with Simulacrum in the same way as the real machine. This provides a safe, convenient way for new operators to learn how to operate the machine, even when the real machine is not available. Some members of the accelerator operations group at SLAC are in the process of creating training scenario scripts that launch an instance of Simulacrum, then set up initial conditions for the scenario.

There is also a plan to use the simulator for community outreach. A simplified graphical user interface has being created to give the general public a "control room operator experience" during SLAC's Community Day. Groups will perform two of the most common LCLS operations tasks (beam steering and FEL tuning), get scored, and can compete for a place on a high score board.

ACKNOWLEDGMENTS

The authors thank Greg White and Timothy Maxwell at SLAC for the initial conversations that inspired Simulacrum, Christopher Mayes (SLAC) and David Sagan (Cornell) for their help with Tao and BMAD, as well as Daniel Allan (BNL), Thomas Caswell (BNL), and Ken Lauer (SLAC) for their help with caproto.

REFERENCES

[1] B. E. Granger and M. Ragan-Kelley. (2019). PyZMQ Documentation, https://pyzmq.readthedocs.io

[2] The ZeroMQ Authors. (2019). ZeroMQ, https://zeromq.org

[3] D. Sagan, "Bmad: A relativistic charged particle simulation library," *Nucl. Instrum. Meth.*, vol. A558, no. 1, pp. 356–359, 2006, Proceedings of the 8th International Computational Accelerator Physics Conference, ISSN: 0168-9002. doi:10.1016/j.nima.2005.11.001

[4] M. Davidsaver. (2019). PVAccess for Python (P4P), https://mdavidsaver.github.io/p4p/

[5] D. Allan. (2019). caproto: a pure-Python Channel Access protocol library, https://caproto.github.io/caproto/index.html

[6] S. Tomin *et. al*, "Progress in Automatic Software-Based Optimization of Accelerator Performance," *Proceedings of IPAC 2016,* Busan, Korea, May 8-13, 2016, doi:10.18429/JACoW-IPAC2016-WEPOY036

[7] H-D. Nuhn *et. al*, "LCLS Undulator Commissioning, Alignment, and Performance," *Proceedings of FEL 2009,* Liverpool, UK, August 23-28, 2009, paper THOA02, pp.714-721.