ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-TUPRB084

HIGH LEVEL SOFTWARE DEVELOPMENT FRAMEWORK AND ACTIVITIES ON VELA/CLARA

D. J. Scott, #,[1] A. D. Brynes,[1] M. P. King,[1] STFC ASTeC, Daresbury Laboratory, United Kingdom
[1]also at The Cockcroft Institute, Warrington, WA4 4AD, United Kingdom

Abstract

The success of modern particle accelerators depends on good high level software. Over the past few years an integrated framework has been developed to better connect machine physicists to VELA/CLARA at the STFC's Daresbury laboratory. This framework is comprised of a number of tools, including, a C++/Python API to interface to the EPICS control system with which all high level software can be developed. The API is encapsulated, extensible and designed to grow as further phases of CLARA are installed. The API is seamlessly integrated with the VELA/CLARA virtual accelerator and other activities by the simulations group. As well as presenting the design choices and methodology we will give an overview of the first control room applications built using our tools and how they will form the basis for a new programme of machine learning and optimisation on CLARA.

INTRODUCTION & OVERVIEW

VELA/CLARA [1, 2] are complementary, compact electron linear accelerators at the STFC's Daresbury Laboratory, designed as an R&D machine for novel FEL schemes and as an underpinning technology demonstrator for a future UK-XFEL, a large scale national facility. CLARA also delivers beam for user exploitation experiments ranging from novel acceleration to efficacy tests for cancer therapies. Optimised operation of machines such as CLARA is non-trivial. There are many non-linear processes and complex dynamics that must be mitigated in order to achieve best outcomes. Due to the different applications of CLARA there is also a requirement to provide flexible set-ups with automated optimisation techniques being desirable. There are also many new and emerging techniques in machine learning and intelligent controls that are yet to be fully applied to these challenges [3-5]. Over the past 5 years a network of software tools, protocols and data has been developed with the aim of integrating the design, simulation, commissioning, characterisation, operation and optimisation of CLARA: CLARA-NET. CLARA is an ideal machine for this development work as:

It is being built in phases, and each phase has a design, installation, commissioning and operation stage, allowing subsequent phases to learn from previous ones. This gives chances to test new ideas and iterate in a timely manner.

It is an R&D machine with a significant proportion of operation time dedicated to Machine Development and prototyping solutions.

CLARA has a role to provide beam for underpinning technology demonstrations for the future UK-XFEL (e.g. cavity BPM, RF structures). CLARA-NET forms a core part of that technology demonstration.

The required operational flexibility with diverse exploitation programme makes an ideal proving ground for new methods aiming to provide this flexibility.

The scope of CLARA-NET is too large to be discussed here, instead the main components and the workflows used for High Level Software Development will be presented.

The Virtual Accelerator (VA)

The VA is a digital copy of the Physical Accelerator (PA) [6]. It is comprised of an Online Model (accurate simulation model of the physical machine [7]), a Virtual Control System and Virtual Hardware. The VA relies on various Data Stores, e.g. previous simulation results and the Master Lattice, (a mark-up language repository of all 'offline data' such as element specification, measured performance, lattice positions, controls variables, etc.). All these parts are connected via Python scripts that provide a common user interface. The VA has a Simulation Framework that can simulate the beam dynamics using a suite of codes, with settings applied either directly, or read from the Virtual or Physical Control System. Results from the simulation are added to the Data Store and can also be written to the Virtual or Real Control Systems. (Within CLARA-NET switching between the PA or VA is as simple as setting a flag in user-code and therefore this distinction need not be explicitly stated in what follows.) Used in this way it is easy to see how the VA can be an invaluable tool for designing High Level Applications, test optimisations (in the virtual or physical space) and gives access and insight to 'hidden' parameters such as the 6D-emittance of the beam. Figure 1 gives a schematic of example workflows using the VA.

High Level Application Development Tools

A 'High Level Application' (HLA) is an automated set of procedures/protocols to perform a specific task on the machine (e.g. a beam measurement). Two of the aims of this part of CLARA-NET are to; encourage new developers from a pool of staff with little application development experience; set common standards of design, tools and workflow such that different developers can take over existing applications with minimum overhead. To achieve this there are a number of preferred tools used by all HLA. Agile project management tools, such as Github [8], Trello [9] and Slack [10] provide open access to all stages of application lifecycle. HLA are written in Python, using a small set of libraries, including PyQT [11] and NumPy [12]. All HLA make use of a common C++/Python middle-level-interface to the control system, CATAP: '*Controls Abstraction To*

#duncan.scott@stfc.ac.uk

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-TUPRB084

Accelerator Physics' (explained in more detail below). To enforce standards software should pass acceptance and quality assurance tests before being allowed in the release environment.

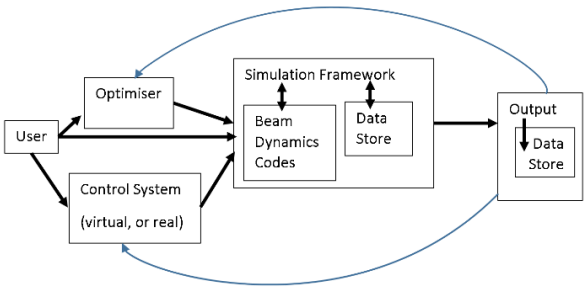

Figure 1: VA workflow schematic.

CATAP: THE MID-LEVEL-INTERFACE

CATAP is a software library containing multiple modules that provides easy read/write access to the main types of accelerator hardware through the control system. In addition to basic control and data reading, standard procedures, analysis techniques etc. are also included. CATAP's main design goals are to be; easy to use, extensible and able to perform any conceivable application. Some of the most important design choices and the advantages they bring are given below.

Ease of use: Each module has *human readable* functions and variable names. End-users should only have to create a single object and everything else automatically follows. Once imported and initialised CATAP abstracts away all the configuration, connections and communications with the Controls System. This gives end users access to function such as:

```
Switch-on(ALL_MAGNETS), quad1.set_current = 1.3
open(SHUTTER_1), gun_rf.set_phase = -15
```

Using this extended toolset it is possible to build '*any conceivable beam experiment*,' all written at a relatively high level that allows developers to concentrate on the experimental procedure, not on the intricacies of controlling and monitoring multiple hardware types.

Python: The source code is written in C++ (so easily included in any C++ application) and has also been compiled into Python modules using boost [13] that can imported into any Python script. Python is a very easy to use scripting language that can be used by novices and experts, thereby widening the pool of possible developers.

An object orientated approach with encapsulation: CATAP creates containers of Virtual Hardware Objects that represent the data associated with each hardware type. Online data is automatically updated from the Controls System. Offline data is read from the Master Lattice. Each hardware type has relevant methods, for example: a magnet object knows how to switch on/off, set a field, degauss etc. a magnet; a BPM object knows how to calculate the beam charge and position. Using this approach it is possible to aggregate hardware types into logical sub-system groupings. For example, the CLARA Photo-Injector module groups together the relevant individual modules required to control and monitor the system, e.g. the Virtual Hardware for the Photo-injector laser position and intensity, virtual cathode camera, camera-images and image analysis, photo-injector laser transport shutters and valves etc. In this way different systems of multiple hardware types can easily be created.

Extensible with ease of maintenance/management: The Virtual Hardware Objects are dynamically instantiated at run-time after reading the Master Lattice, as new hardware comes online, or is upgraded all that is required is an update to the Master Lattice for these changes to be propagated to all users of CATAP.

Shared solutions: Simple procedures are contained within CATAP. This means writing procedures for degaussing, switching on the RF system, etc. are problems that only need solving once. As solutions become accepted and robustly tested they can be moved down the toolchain. For example a cathode charge scan was developed as a Python application enabling quick prototyping and testing of different methods. When the procedure was agreed it was then implemented at the C++ level, making it available to all users of CATAP. A next stage could then be to implement the procedure within the Controls System, which would be even more robust.

External user exploitation libraries: As the code is compiled into python modules from c++ source it is very simple to create managed, external user versions of the CATAP library. These libraries will have stricter limits on the control of hardware suitable for external users, whilst also giving them the ability to monitor all signals and design their data acquisition accordingly.

Example

Figure 2 shows a simple python script that creates a magnet interface and performs some simple operations and then shows the workflow from the C++ side with the following numbered items:

1. hardware_complex.setup() specifies the type of hardware to search for in the Master Lattice and returns a container of parameters for each object
2. A Hardware class is created with hardware type, machine area, and a container for the specific component parameters (magnet parameters in this case) and this object is held by hardware_complex
3. The hardware type of the component checked (magnet) then the specific factory (magnet factory) is invoked to setup the component using the specific component parameters
4. Each component will then establish a connection to the Controls System via its own Interface so we can interact with the physical/virtual hardware.
5. (Not in diagram) once this specific component object is created, it is added to a container owned by Hardware Factory. The component is accessed via the object interface, or through getter/setter functions using the components unique name or any associated alias.

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-TUPRB084

```
# import CATAP hardware module
>>from CATAP import hardware_complex
# create an interface to the physical
# accelerator magnets
>>hardware_complex.setup('MAGNETS','PHYSICAL')
# switch on all magnets
# (returns success value)
>>hardware_complex.switchOn('All_MAGNETS')
True
# get quad-03 magnet object
>>mag = hardware_complex.get_magnet('QUAD-03')
# set magnet current via method
# (returns success value)
>>mag.set_current(1.0)
True
# set magnet current via assign
# (returns success value)
>>mag.current = 3.0
True
# get magnet current via method
>>mag.get_current()
3.0
```

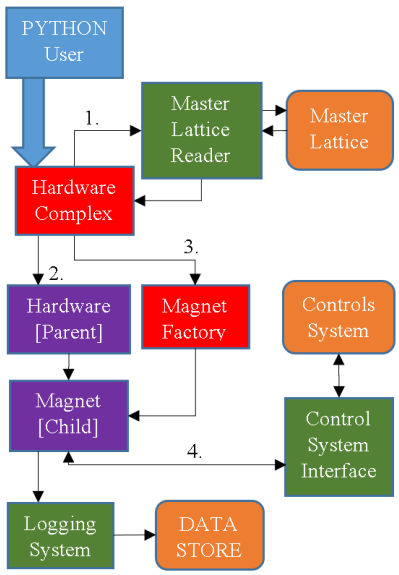

Figure 2: (Top) example python script creating and using magnets on CLARA. (Bottom) flow chart of the main C++ classes structure (square boxes) and how they interact with external systems (orange, rounded edge boxes). Similar colours show grouping by type.

Example Applications

During the last run of CLARA [14] CATAP as a high level application development tool was used for the first time *in the wild*. The number of application developers was increased from effectively zero to twelve and the number of applications worked on was ~40. Applications were developed for a myriad of purposes: online image analysis, diagnostic calibration, beam steering, parameter scans such as charge on the cathode, measuring the momentum, momentum spread, emittance and RF phase, setting up the laser beam on the cathode etc. Below are specific examples of how the system was used that highlight the potential of the system. It must be stressed *these applications were made by developers who had no previous experience building HLA*.

BPM Calibration and Signal Tagging

This application routinely calibrated the BPMs to charge measurement devices. During operations it was found that spurious, false-positive, BPM position readings could be generated due to beam scraping in upstream components even though there was no signal on an adjacent screen. Once further investigated it was found that the low level BPM voltages could be used to tag these false-positives with a warning not to be trusted. This meta-data was then immediately available to all applications using CATAP.

Align-On-BPMs

This application used the BPM and magnet modules to set and optimise the orbit. The entire application, including steering methods, was prototyped using the VA. When the false-positive BPM data was identified and the changes made to CATAP these changes were immediately available to this application and it was updated with minimal effort.

Quick-Spectra

This is an example of 'agile application creation and development' during shift. During a user experiment it became apparent that a quick projection of the beam spectra including averaging, comparison with reference data and automated FWHM estimation would significantly decrease set-up time. Using CATAP a solution was built in a few hours that was immediately useful. As we are working within a common framework it was possible for other operators to make slight improvements to this HLA on subsequent shifts, directly responding to further user requests.

Machine learning

As an example, an automated unmanned RF conditioning procedure has been built with this toolset [15]. It requires a breakdown event to be detected in the RF power traces. This event detection routine was first prototyped in Python, then, when ready, pushed down to CATAP. Concurrent to this, new methods of event detection using Neural Networks [16] to classify and characterise events were also prototyped at the Python Level using CATAP. As these methods are proved the systems are designed to make it trivial to switch over from the conventional methods to the new.

CONCLUSIONS AND FUTURE WORK

The development activities presented here are one part of the wider CLARA–NET. Working in concert they will enable ever increasing control and optimisation of the VELA/CLARA machines by strengthening the links between simulations and operations and enabling much HLA development independent of beam-time. An integrated approach to High level Software combining operations and simulations is a powerful tool that will bring many benefits. Perhaps there could be wider collaborations to work on shared solutions that all can benefit from?

REFERENCES

[1] P. A. McIntosh *et al.*, "VELA: a new accelerator technology development platform for industry", in *Proc. IPAC'14*, Dresden, Germany, June 2014, pp. 2471-2473. doi:10.18429/JACoW-IPAC2014-WEPME083

[2] D. Angal-Kalinin *et al.*, "Commissioning of front end of CLARA facility at Daresbury laboratory", in *Proc. IPAC'18*, Vancouver, BC, Canada, May 2018, pp. 4426. doi:10.18429/JACoW-IPAC2018-THPMK059

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-TUPRB084

[3] Intelligent controls for particle accelerators, https://www.cockcroft.ac.uk/events/ICPA/

[4] Machine Learning Applications for Particle Accelerators, https://conf.slac.stanford.edu/icfa-ml-2018/

[5] 2nd ICFA Workshop on Machine Learning for Charged Particle Accelerators, https://indico.psi.ch/event/6698/contributions/speakers

[6] T. J. Price *et al.*, "Virtual VELA-CLARA: the development of a virtual accelerator" , in *Proc. IPAC'18*, Vancouver, BC, Canada, Apr.-May 2018, pp. 4773-4776. doi:10.18429/JACoW-IPAC2018-THPML060

[7] M. S. Toplis *et al.*, "Comparison of model vs. reality for VELA", presented at the 6th Int. Particle Accelerator Conf. (IPAC'16), Busan, Korea, May. 2016, paper TUPOW028, unpublished.

[8] Github, https://www.github.com

[9] Trello, https://www.trello.com

[10] Slack, https://www.slack.com

[11] PyQT, https://wiki.python.org/moin/PyQt

[12] NumPy, https://wiki.python.org/moin/NumPy

[13] Boost, https://www.boost.org

[14] D. Angal-Kalinin *et al.*," Status of CLARA front end commissioning and first user experiments", presented at the 10th Int. Particle Accelerator Conf. (IPAC'19), Melbourne, Australia, May 2019, paper TUPRB083, this conference.

[15] L.S. Cowie *et al.* "RF Conditioning o fthe CLARA 400 Hz Photo-injector", presented at the 10th Int. Particle Accelerator Conf. (IPAC'19), Melbourne, Australia, May 2019, paper TUOTS065, this conference.

[16] D. J. Scott 'Artificial Neural Networks 1: An Introduction, Toy Model and CLARA Linac Breakdown Recognition', ASTeC, Daresbury United Kingdom, Rep. VELA-EN-20170814, June 2017.