ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA017

pyAT, Pytac AND pythonSoftIoc: A PURE PYTHON VIRTUAL ACCELERATOR

W. Rogers*, T. J. R. Nicholls, A. A. Wilson, Diamond Light Source, Oxfordshire, UK

Abstract

Virtual accelerators are used for testing control system software against realistic accelerator simulations. Previous virtual accelerators for synchrotron light sources have used Tracy and Elegant as the simulator, but without Python bindings for accelerator simulations it has been difficult to create a virtual accelerator using Python. With the development of Python Accelerator Toolbox (pyAT), that is now possible. This paper describes the combination of pyAT, Python Toolkit for Accelerator Controls (Pytac) and pythonSoftIoc to create an EPICS-based virtual accelerator for Diamond Light Source.

MOTIVATION

High-level control system software is designed to interact with the control system that is connected to the real hardware. Testing this software can be inconvenient for two main reasons: during design and commissioning the hardware might not yet exist, and during its operational lifetime the hardware is in use most of the time.

A *virtual accelerator* is an application that allows testing a control system by providing the same interface as a subset of the control system that is required for the operation of high-level applications. Although it is possible to provide dummy values for different control system parameters, it is much more useful to combine those parameters with a simulation so that they respond in a physically accurate way to any changes. The software under test then requires no changes in order to run against a virtual accelerator.

Python has a number of advantages for developing a virtual accelerator: it is free and open source, is very widely used in both science and industry, has many useful third-party libraries available, is simple to start using and is capable of building large applications that scale well.

To build this virtual accelerator we needed a number of components:

- a simulation code for synchrotron light sources that can be called from Python
- a Python framework that understands the elements of a particle accelerator
- the ability to convert between engineering units (used in the control system) and physics units (used in simulation codes)
- a server that can hook the Python code into the control system

The following sections describe these components.

* will.rogers@diamond.ac.uk

pyAT

Accelerator Toolbox (AT) [1] is a simulation code for synchrotron light sources developed for use in Matlab. Its numerical engine is based around 'integrators' that calculate the effect of a particle in 6D phase s pace. For efficiency purposes these integrators were written in C and compiled for use in Matlab.

This design allowed for the same integrators to be compiled for use in Python code. pyAT uses the same numerical engine as AT, with Python classes and functions written to derive accelerator parameters from the engine. It uses the libraries NumPy and SciPy for several numerical utilities. Previous virtual accelerators for synchrotron light sources have used Tracy [2] [3] and Elegant [4] [5] as the simulator, but these were missing Python bindings.

pyAT provides a number of features. Lattice and element types are defined and may be loaded and saved to different file formats. It performs particle tracking and allows calculation of transfer matrices and closed orbit with radiation included or excluded. Other derived parameters that are calculated include linear optics, radiation integrals and detail of the beam envelope. It includes a plotting package that uses Matplotlib to provide various plotting functions.

Testing has shown pyAT to give exactly the same numerical results as AT with a speed comparable to other accelerator simulations [6].

Pytac

Python Toolkit for Accelerator Controls (Pytac) is a Python library designed to enable working with the different parts of accelerators. Each element in an accelerator is represented by an object that may have one or more 'fields' corresponding to physical parameters. The ability to address the different elements of an accelerator by position in the accelerator and element family is often useful in high-level applications and Python scripts.

```
>>> bpm1 = lattice.get_elements('BPM')[0]
>>> bpm1.fields()[pytac.LIVE]
['enabled', 'x', 'y']
>>> bpm1.get_value('x', data_source=pytac.LIVE)
-2.1e-5
```

Pytac allows requesting the live values of the parameters from the accelerator control system. It also allows efficiently requesting the values for an entire family.

```
>>> lattice.get_element_values('BPM', 'x')
[-4.6e-05, 8.2e-05, 7e-05, ...]
```

Many of the ideas in Pytac were inspired by a similar application Matlab Middle Layer (MML) [7].

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA017

Unit Conversion

It is often useful to use the same accelerator parameter in different unit systems: the units used by physicists for calculations, and the units used by the control system for control and monitoring. Pytac has a built-in unit conversion mechanism that allows requesting and setting parameters in either of these unit systems. The following example requests a quadrupole setting in engineering units (A) and physics units (Tm^{-1}):

```
>>> quad.get_value('b1', units=pytac.ENG)
103.18108367919922
>>> quad.get_value('b1', units=pytac.PHYS)
-1.0192934647760261
```

Two unit conversion mechanisms are currently available: polynomial (often used for linear conversions) and piecewise cubic hermite interpolating polynomial (PCHIP), an algorithm provided by SciPy [8] that allows smoothed interpolation between arbitrary measured data points. Implementing further unit conversion mechanisms is simple.

Configuration

The accelerator definition is stored in a number of CSV files. These are easy to edit and are efficiently stored in the Git version control system. The configurations for the Diamond accelerators are exported using a Matlab script from the existing configurations that are set up in MML.

ATIP

Accelerator Toolbox Interface for Pytac (ATIP) is the adapter that makes pyAT available as a simulator for Pytac. Once loaded, pyAT provides an online model for interactive use in Pytac.

```
>>> lattice.set_default_data_source(pytac.SIM)
>>> lattice.get_element_values('BPM', 'x')
[0.0, 0.0, 0.0, ...]
>>> h_corr = lattice.get_elements('HSTR')[0]
>>> h_corr.set_value('x_kick', 0.1, units=pytac.ENG)
>>> lattice.get_element_values('BPM', 'x')
[0.24630504031808942, 0.12495575893699563,
 -0.1257213016476168, ...]
```

ATIP has a simple asynchronous threading mechanism. Any changes that would require a recalculation using pyAT are placed on a queue. A dedicated simulation thread loops continuously, checking whether there are any items in the queue. If so, it empties the queue, applies each change to its model and recalculates. When a request for data is received, ATIP will check whether an update is pending and if so will wait until the recalculation is complete before returning that data.

An outline of how ATIP integrates pyAT into Pytac is shown in Fig. 1.

pythonSoftIoc

At Diamond Light Source we use the EPICS distributed control system, with many servers known as IOCs that provide some subset of the control system parameters (process variables, or PVs). Many IOCs are embedded devices that interact directly with hardware, but often it is useful to make standalone IOCs that may not represent hardware devices at all; these are called *soft IOCs*. pythonSoftIoc [9] is a Python library that allows creating an EPICS IOC using only Python code. The virtual accelerator uses this library to create an IOC using the PV names used on the Diamond accelerator, then respond appropriately to any interactions.

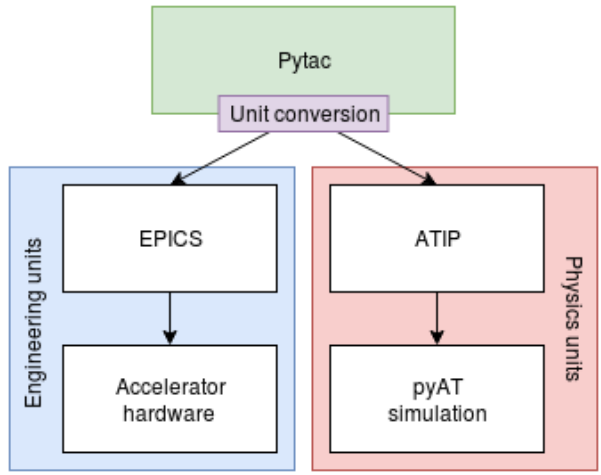

Figure 1: How ATIP integrates into Pytac.

A PURE PYTHON VIRTUAL ACCELERATOR

Using the tools above, it is now possible to assemble a virtual accelerator using only Python. The definition of the accelerator is provided by Pytac, the accelerator simulation is provided by pyAT, and the EPICS IOC is provided by pythonSoftIoc: see Fig. 2.

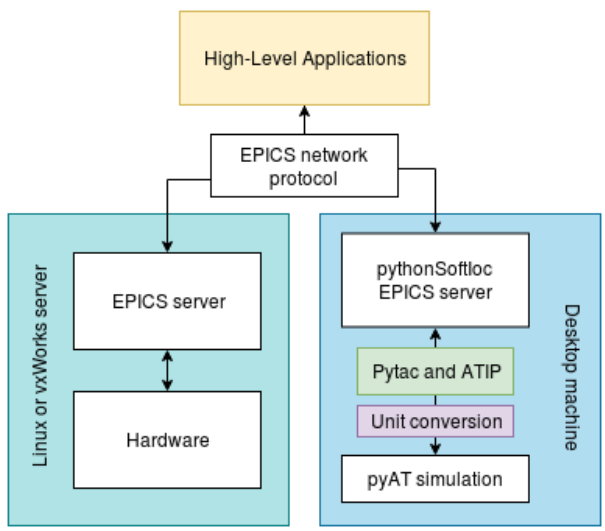

Figure 2: How high-level applications may use either the control system (left) or the virtual accelerator (right) interchangeably.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA017

This virtual accelerator is now being used to test the following high-level applications at Diamond.

High-Level Applications

Slow Orbit Feedback The slow orbit feedback system is well simulated using the virtual accelerator. The beam position is simulated and returned via the BPM elements; the feedback system calculates a correction to be applied to the corrector magnets over EPICS and the simulation is updated once the correction has been applied. The 1 Hz rate of the slow feedback system can be handled by the virtual accelerator.

RF Feedback The orbit feedback systems correct the beam orbit so that the BPM readings are zero. However, there are multiple solutions to this, some of which may have forced the electrons to a different energy than designed. The RF feedback system determines whether the net effect of the corrector magnets includes such a change; if so, it changes the RF frequency to remove it. The slow orbit feedback and RF feedback systems work together, and can both be tested against the virtual accelerator.

Tune Feedback Diamond's tune feedback system [10] uses a subset of the quadrupoles to correct variations in the tunes. A response matrix dictates how the tunes change when the quadrupole settings change; this matrix is inverted to determine the quadrupole changes required to correct a deviation in the vertical and horizontal tunes.

Vertical Emittance Feedback Diamond runs a vertical emittance feedback system [10] that keeps the beam size broadly constant. Since pyAT provides the emittance value for the ring using the Ohmi Envelope formalism, it is possible to use the virtual accelerator to test the vertical emittance feedback system.

BURT The Back Up and Restore Tool (BURT) is used at Diamond to save and restore machine configuration in the form of stored values for specific PVs. We have used the virtual accelerator to test BURT functionality as we develop a new version of Burt in Python. Burt may also be used to save different configurations of the virtual accelerator.

Challenges

There are a number of details of the high level applications that make using the virtual accelerator challenging.

Speed of calculation and update rates A fundamental limitation of this design of virtual accelerator is the elapsed time taken in order to recalculate machine parameters. These simulations are typically run on desktop machines.

The simulation used for the virtual accelerator has the two parts summarised in Table 1. This gives an approximate update rate of 1 Hz, sufficient for most of the high-level applications.

The one example for which this caused problems was the vertical emittance feedback system. The PV that provides the vertical emittance value updates at about 5 Hz, and the feedback system reports slower updates as a failure. As Python is inherently single-threaded it is impossible to update a PV on the virtual accelerator while the recalculation is taking place; in any case, unless there are valid new values to report more frequent updates may cause feedback systems to misbehave.

The vertical emittance feedback system can be tested against the virtual accelerator if the update check is disabled; whether this check is useful in the application itself is being considered.

Table 1: Virtual Accelerator Simulation Functions Executed on a Typical Desktop PC

Function	Description	Runtime
`linopt()`	Derive linear optics	0.25 s
`ohmi_envelope()`	Calculate emittance	0.65 s

Control system complexities Sometimes the EPICS Control System is more complicated than the simple view presented by a virtual accelerator.

One example at Diamond is the way that the quadrupoles are controlled, which uses a number of PVs to aggregate contributions to the magnet setpoint from different sources, one of which is the tune feedback system [10]. In this case it was possible to test the tune feedback system by providing 'mirror' PVs that respond in the same way to the original setpoint PVs; in other cases simple transforms can be applied to the PV value.

Certain PVs at Diamond are available as individual PVs per element but are also available for convenience as a waveform PV: for example, there are two waveform PVs containing all 173 BPM values in the horizontal and vertical planes. The natural way for the virtual accelerator to provide this information is per-element, but applications are more likely to use the waveforms. To solve this we added a configuration mechanism to aggregate individual values into addtional waveform PVs.

These transformations are handled by classes in ATIP, and it would be possible to make other transformations by writing similar classes.

NOTES ON SOFTWARE

All of the components described above are open source and the source code is available on Github. pyAT, Pytac and ATIP are available on the Python Package Index (PyPI) [11] for simple installation using pip. Any interest in using or collaborating on these projects would be welcomed.

Jupyter notebooks are a good way of demonstrating the capabilities of these applications. Some example notebooks exist in the Git repositories, and further examples are being developed.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA017

Python versions 2.7 and 3.5+ are supported, although Python 2 support will be deprecated in the near future in line with the approach of the rest of the Python community.

FURTHER WORK

The components described above begin to form a versatile toolkit for accelerator physics and control system applications using Python.

The next application in this toolkit is Visualiser and Optimiser for Linear Optics (Volo), which uses ATIP and pyAT to allow interactive lattice viewing and editing via a PyQt GUI. Most of the capability for this tool exists in the components described above, meaning that the Volo project mostly requires using these components and constructing an intuitive user interace. A prototype of this application is under development; a screenshot of an early version is shown in Fig. 3.

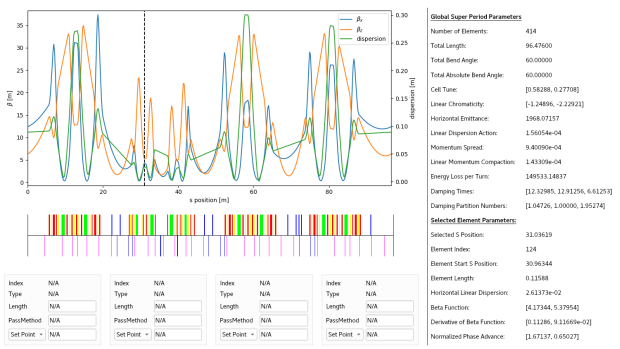

Figure 3: A screenshot from an early version of Volo.

REFERENCES

[1] A. Terebilo, "Accelerator Toolbox for MATLAB", SLAC-PUB-8732, 2001.

[2] M. Boge, "Update on TRACY-2 Documentation", SLS Internal Note, SLS-TME-TA-1999-0002 (1999).

[3] M. T. Heron *et al.*, "The Diamond Light Source Control System", in *Proc. EPAC'06*, Edinburgh, UK, Jun. 2006, paper THPCH113, pp. 3068–3070.

[4] M. Borland, "elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation," Advanced Photon Source LS-287, September 2000. doi:10.2172/761286

[5] P. P. Goryl, A. I. Wawrzyniak, M. Sjöström, and T. Szymocha, "An Implementation of the Virtual Accelerator in the Tango Control System", in *Proc. ICAP'12*, Rostock-Warnemunde, Germany, Aug. 2012, paper MOSBC3, pp. 23–25.

[6] W. A. H. Rogers, N. Carmignani, L. Farvacque, and B. Nash, "pyAT: A Python Build of Accelerator Toolbox", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 3855–3857. doi:10.18429/JACoW-IPAC2017-THPAB060

[7] G. J. Portmann, W. J. Corbett, and A. Terebilo, "An Accelerator Control Middle Layer Using Matlab", in *Proc. PAC'05*, Knoxville, TN, USA, May 2005, paper FPAT077, pp. 4009–4011.

[8] SciPy documentation for PCHIP, https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.interpolate.PchipInterpolator.html

[9] pythonSoftIOC on Github, https://github.com/Araneidae/pythonIoc

[10] M. T. Heron *et al.*, "Feed-forward and Feedback Schemes applied to the Diamond Light Source Storage Ring", in *Proc. IPAC'14*, Dresden, Germany, Jun. 2014, pp. 1757–1759. doi:10.18429/JACoW-IPAC2014-TUPRI081

[11] PyPI, https://pypi.python.org/pypi