ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-MOPB088

EXPLORATORY TESTS FOR THE DESIGN OF A PYTHON ACCELERATOR MIDDLE LAYER

S. M. Liuzzo*, J.-L. Pons, S. White, ESRF, Grenoble, France
A. Al-Sakeeri, M. Apollonio, R. Nieuwenhuis, R. Rocca, MAX IV Laboratory, Lund, Sweden
M. B. Alves, F. H. de Sá, M. M. S. Velloso,
Brazilian Synchrotron Light Laboratory, LNLS, Campinas, Brazil
M. Gaughran, T. Nicholls, Diamond Light Source, Oxfordshire, UK
V. Gubaidulin, Synchrotron SOLEIL, Saint-Aubin, France
Y. Hidaka, NSLS-II, Brookhaven National Laboratory, Upton, NY 11973, USA
T. Olsson, W. Sulaiman Khail, Helmholtz-Zentrum Berlin, Berlin, Germany

Abstract

Several laboratories and facilities recently started joined efforts towards the realization of a python Accelerator Middle Layer (`pyAML`) for design, commissioning and operation of accelerators. This software is intended as a successor to Matlab Middle Layer (`MML`), inheriting its features but also expanding to new ones (e.g., nonlinear optics and machine learning tools). Presently, several codes are available that provide some of the desired features. These codes have been adapted and tested at several participating laboratories to give input to the design of the `pyAML`. The most relevant features and results have been analyzed and are presented here together with the implications for the `pyAML` design.

INTRODUCTION

Presently, the Matlab Middle Layer (`MML`) software [1] is used by most synchrotron light source laboratories. The `MML` software provides: 1) independence from the underlying control system, 2) abstracted element naming conventions, 3) graphical interfaces for common tuning and design tools and 4) the possibility to run measurements and control loops in "simulation mode", thus allowing software testing without the need for costly beam time, or in absence of the accelerator for new projects (thus acting as a virtual accelerator). For laboratories using `MML`, the tools developed in one laboratory are immediately usable by all other laboratories, thus fostering exchanges and collaboration between accelerator physicists around the world. However, over time the `MML` software has branched into many versions. Updating `MML` would require an entire re-writing of its core and user interfaces in `Matlab` [2], a proprietary software poorly known among students and young professionals entering the field of particle accelerators today. `Python` [3], on the other hand, is open-source, among the most widely used programming languages, and benefits from extensive open-source scientific libraries. Selecting `Python` as the software language for the replacement of `MML` allows us to reach a much wider user base, simplifies interfaces to other software libraries used by our community and will help future developments critical for the new generation of accelerators. Since 2017, AT (`pyAT`) [4], `Xsuite` [5] and laboratory-specific tracking codes already exist in a `Python` version.

EXPLORATORY TESTS

A large community of accelerator physicists and software engineers has started to collaborate on the realization of a Python Middle Layer (`pyAML`) [6]. As a first step, different codes have been analyzed, configured for some test cases and run on existing accelerators. The codes considered are: `MML` [1, 7], `Pytac` [8], `Pytac + Bluesky` [9, 10], `Ophyd + Bluesky`, `pyACAL` [11] and `PAMILA` [12]. `MML` is using `Matlab`, while all other codes are implemented in `Python`.

MML `MML` is the reference code. Most laboratories in the collaboration use it. No further tests were done, but the user-friendliness and the principles for how to perform a measurement were compared.

Pytac `Pytac` (Python Toolkit for Accelerator Controls) is developed and used at Diamond Light Source (EPICS control system). The configuration and magnet calibration curves are stored as CSV files and can be exported from an existing `MML` configuration or written from scratch. `Pytac` can also be connected to a simulator, `ATIP` [13]. The code was tested for the ESRF-EBS (TANGO control system) and BESSY II (EPICS control system) storage rings [14]. For both machines, the creation of groups of magnets (families) was straightforward. Interfacing to the TANGO control system required minimal modifications to the code. For BESSY II, the configuration of calibration curves was also tested but it was found to be confusing and not so easy to modify. The `Pytac` scripting language was as intuitive as in `MML` and allowed writing a script for a orbit response matrix measurement (ORM) in a rather short time. However, the user needs to implement all the steps themselves. `Pytac` is only a hardware abstraction layer and does not include features for experimental control and data acquisition. For ESRF-EBS, the ORM script was debugged using the ESRF virtual control system [15] and the measurement could later run with real beam without any major modifications. For BESSY II, the script was developed using the `ATIP` simulator which worked well, and could also run on the real machine without any major modifications. Figure 1 shows

* simone.liuzzo@esrf.fr

ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-MOPB088

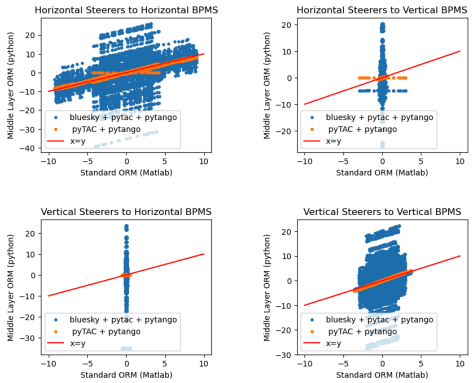

Figure 1: Correlation between orbit response matrix data acquired using `Pytac`, `Pytac + Bluesky` or the existing ESRF measurement system. One faulty horizontal BPM shows a discrepancy for both measurements.

good agreement between the measured orbit response matrix, done either using `Pytac` or existing in-house ESRF tuning tools to pilot the magnets and acquire the data.

Bluesky `Bluesky` is a collection of Python libraries for experimental control and data acquisition. Testing using only components in the `Bluesky` ecosystem was done at BESSY II. The `Ophyd` package [16] was used for hardware abstraction. For beamlines, ready-to-use devices exists but for accelerator devices they had to be implemented. It was easy to write simple devices but complex ones required a lot of work. It was also not easy to understand how to control the settling time of the devices and there seemed to be no natural way to group devices. `Bluesky` was then used to run the measurement. The concept of measurement plans was an improvement compared to `MML` because it allowed for separation between measurement procedure and configuration and data storage steps. The measurement engine was also an improvement because it automatically added metadata in a standardized format to the measurement results. It also allowed subscribing to the output and easily add features like live output and live plotting. Finally, `Databroker` [9] was tested to access the measurement data stored in a database but this was found to be difficult for a user used to browse among files. Since the tests, a new package, `Tiled`, has been developed, which might solve these issues.

At the time of the tests, no hardware abstraction interface existed for TANGO and an intermediate level based on `Pytac` was implemented to test `Bluesky` at ESRF-EBS[1]. The `Bluesky` measurement orchestrator was used to run the ORM measurement. About 5 times more lines of code were needed compared to the simple `Pytac` scripting implementation of the measurement. The data recovery was not straightforward, resulting in an invalid ORM measurement (as visible in Fig. 1) when running on the real machine compared to the results obtained when running within the ESRF virtual control system [15]. Also, the `Bluesky` orchestration separated thread does not copy the environment variables of the original thread, which resulted in unwanted action on the real beam.

pyACAL `pyACAL` (Python Accelerator Control Abstraction Library) was mainly developed at the Brazilian Synchrotron Light Laboratory (LNLS) as an attempt to make the vast experiment control code developed for SIRIUS (EPICS control system) in the last ten years usable for other accelerators. The library has an interface to abstract the control system type: EPICS or TANGO (the TANGO interface was developed at ESRF); and the simulation tool, such as pyAT and PyAccel [17], which can easily be expanded for other codes. `pyACAL` allows complex unit conversions from hardware to physics, and vice versa. Communication with control system equipment is made via Python objects called *devices* that encapsulate the main features of common accelerator subsystems, such as BPMs, magnets, DCCTs, LLRFs. In a *device* object, setter and getter attribute methods are used to read and control the parameters, embedding standard procedures such as "set and wait", "set in steps", etc. Device objects are the building blocks of the experiment classes, whose development and structure are considerably simplified by having these standard procedures already handled within devices. This infrastructure allowed to easily create libraries for BBA, slow orbit correction, response matrix measurement, and chromaticity and dispersion measurements. All were successfully tested in the SIRIUS storage ring. For the TANGO interface, the initial configuration and set-up were done in a few days, the main issue was the absence of the concept of family/group of devices at the control system interface level since it was based on EPICS. This feature is available in TANGO and a workaround to mimic the behavior of EPICS PVs by connecting to individual lower-level TANGO device servers was implemented. The `pyACAL` ORM, chromaticity, tune and dispersion applications were then tested at several TANGO facilities (ESRF-EBS, SOLEIL and both MAX IV rings) successfully. Figure 2 shows a chromaticity measurement at MAX IV. `pyACAL` was also briefly tested at BESSY II (EPICS) but the interface was not yet sufficiently machine agnostic for it to work. The code was developed in a time span of just two weeks after the 2024 Accelerator Middle Layer workshop. Hence, there was not much thought spent on making the machine configuration setup user-friendly and some assumptions were not general enough for other accelerators. However, these are issues that can easily be solved in future versions of the code.

PAMILA `PAMILA` was recently developed at NSLS-II (EPICS control system) as a testbed for innovative features envisioned for the next-generation accelerator middle layer. One of the key features is its unique ability to handle any device-dependent unit conversions, including multiple-input multiple-output (MIMO) conversions as long as the conversion can be defined as a `Python` function. A current

[1] Since the tests, `ophyd-async` has been developed [16], which supports both EPICS and TANGO, but this has not been tested in the scope of this work.

ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-MOPB088

Table 1: Features of the Tested Codes

Feature	MML	Pytac	Pytac + Bluesky	Ophyd + Bluesky	pyACAL	PAMILA
Control agnostic	yes	easy	difficult	no[a]	yes	almost[b]
Machine agnostic	yes	yes	yes	yes	almost	yes
Access to units	yes	yes	yes	yes	yes	yes
Simulation capabilities	yes	yes	yes	no	yes	yes
Simulation engine agnostic	yes	no	no	no	yes	yes
Easy configuration	almost	yes	yes	almost	no	no
Easy data retrieval	yes	yes	no	no	yes	yes
Existing HLAs[c]	yes	yes[d]	no	no	some	some
User friendly scripting	yes	yes	no	sometimes	yes	yes
Calibrations	basic[e]	basic[e]	basic[e]	no	yes	yes
Grouping of elements	yes	yes	yes	no	yes	yes
Automatic metadata	almost	no	yes	yes	almost	yes
Modular code	almost	yes	yes	yes	yes	yes
Documentation	yes	yes	yes	yes	no	yes
Demos / examples	some	yes	yes	yes	yes	yes
Several machines in one script	no	yes	no	yes	yes	yes
Archive/history features	no[e]	no	no	no	no	no
Digital shadow	no	no	no	no	no	no
Commissioning simulations	no	no	no	no	no	no
HPC integration (CPU/GPU)	no	no	no	no	no	no

[a] available now with ophyd-async but not tested within this paper, [b] if interface for TANGO is implemented, [c] high level applications, [d] Diamond specific, [e] no combined function magnets, available at MAX IV

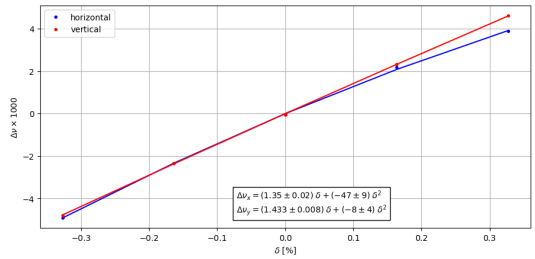

Figure 2: Chromaticity measurement done with pyACAL on the MAX IV 3 GeV ring. The measured chromaticities [1.35, 1.43] agree well with MML [1.33, 1.44].

drawback is the complexity involved in specifying device configurations; further simplification will likely be necessary in future versions. All input and output values of the *get* and *put* methods of any element are pint [18] quantity objects. This ensures consistent and transparent unit handling throughout the package, freeing users from being forced to remember or manually check the exact units associated with the values of interest, and eliminating the need to convert units before or after function calls. It also spares developers from having to re-implement common device-independent unit conversions (e.g., from nm to mm). PAMILA is also designed to integrate seamlessly with the Bluesky ecosystem to take full advantage of modern experimental orchestration and data management technologies. Although it currently supports only EPICS, PAMILA is designed to be control-system agnostic. Supporting another control system simply requires the implementation of classes analogous to Ophyd's "Signal" and "Device" classes. Finally, PAMILA introduces the use of "stage" and "flow" objects to define high-level applications (HLAs). A flow (i.e., sequencer) specifies a se-quence of stages (e.g., "acquire", "postprocess", and "plot") to be run, and can enter and exit at any stage. To run the HLA, a user only needs to call its *run* method without any arguments, as all the options for the HLA are specified in the "Params" object based on pydantic [19] for each stage. This modular approach facilitates re-use of existing HLAs by exposing comprehensive, yet manageable, options to end users. The dispersion-chromaticity measurement HLA was successfully tested both in the live and simulator modes at NSLS-II.

CONCLUSIONS

After testing the above-mentioned codes, it was clear that no code included all features desired for pyAML. The features are listed in Table 1 and marked for each of the codes that have been tested. Key features are control system and machine agnosticism, simulation capabilities and easy configuration and data retrieval. The comparison shows the need for a new common development, building on and inspired by the best practices and development choices of these codes. A prototype version of pyAML is being developed to have several functional use cases by the end of 2025.

ACKNOWLEDGMENT

The authors wish to acknowledge the whole pyAML collaboration. The authors also want to thank P. Schnizer for advice on Bluesky.

ISBN: 978-3-95450-248-6 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2025-MOPB088

REFERENCES

[1] 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), ser. Particle Accelerator Conference, JACoW Publishing, Geneva, Switzerland, pp. 4009–4011. https://jacow.org/p05/papers/FPAT077.pdf

[2] MATLAB, version 7.10.0 (R2010a), The MathWorks Inc. ().

[3] G. van Rossum, B. Warsaw, and N. Coghlan, "Style guide for Python code," PEP 8, 2001. https://www.python.org/dev/peps/pep-0008/

[4] 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), JACoW Publishing, Geneva, Switzerland, pp. 3855–3857. DOI: 10.18429/JACoW-IPAC2017-THPAB060. https://jacow.org/ipac2017/papers/THPAB060.pdf

[5] G. Iadarola *et al.*, "Xsuite: An Integrated Beam Physics Simulation Framework," in *Proc. HB'23*, (Geneva, Switzerland), ser. ICFA Advanced Beam Dynamics Workshop on High-Intensity and High-Brightness Hadron Beams, JACoW Publishing, Geneva, Switzerland, Mar. 2024, pp. 73–80, ISBN: 978-3-95-450253-0. DOI: 10.18429/JACoW-HB2023-TUA2I1.

[6] pyAML collaboration. (), https://github.com/python-accelerator-middle-layer

[7] Matlab Middle Layer. (), https://github.com/atcollab/MML

[8] W. Rogers and R. Vasile, "Python Toolkit for Accelerator Controls". (), https://pytac.readthedocs.io/en/latest/

[9] D. Allan, T. Caswell, S. Campbell, and M. Rakitin, "Bluesky's Ahead: A Multi-Facility Collaboration for an a la Carte Software Project for Data Acquisition and Management," *Synchrotron Radiat. News*, vol. 32, no. 3, pp. 19–22, 2019. DOI: 10.1080/08940886.2019.1608121. https://doi.org/10.1080/08940886.2019.1608121

[10] Bluesky, https://blueskyproject.io/bluesky/main/index.html

[11] F. de Sa *et al.* "Python Accelerator Control Abstraction Library." (2024), https://github.com/python-accelerator-middle-layer/pyacal-test

[12] Y. Hidaka, "Particle Accelerator MIddle LAyer (PAMILA)". (), https://github.com/python-accelerator-middle-layer/pamila

[13] ATIP - Accelerator Toolbox Interface for Pytac. (), https://atip.readthedocs.io/en/latest/

[14] T.Olsson, S.Liuzzo, *et al.*, "Matlab Middle Layer", Accelerator Middle Layer Workshop, DESY, Hamburg, June 2024.

[15] S. M. Liuzzo *et al.*, "The ESRF-EBS Simulator: A Commissioning Booster," in *Proc. ICALEPCS'21*, (Shanghai, China), ser. International Conference on Accelerator and Large Experimental Physics Control Systems, JACoW Publishing, Geneva, Switzerland, Feb. 2022, pp. 132–137, ISBN: 978-3-95-450221-9. DOI: 10.18429/JACoW-ICALEPCS2021-MOPV012. https://jacow.org/icalepcs2021/papers/MOPV012.pdf

[16] Bluesky, "Ophyd Asynch". (), https://blueskyproject.io/ophyd-async/main/index.html

[17] F. de Sa *et al.*, "Python module for beam dynamics tracking and optics calculations". https://github.com/lnls-fac/pyaccel

[18] H. E. Grecco. "Pint". (), https://github.com/hgrecco/pint

[19] S. Colvin *et al.*, "Pydantic", https://github.com/pydantic/pydantic