ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBR03

STATUS OF BLUESKY DEPLOYMENT AT BESSY II*

William Smith†, Sebastian Kazarski, Roland Müller, Luis Vera Ramirez, Pierre Schnizer, Simone Vadilonga
(HZB, Berlin)
Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Berlin, Germany

Abstract

The modernization plan for the experimental DAQ at the BESSY II [1] is underpinned by the capabilities provided by the Bluesky software ecosystem [2]. To interface with the hardware Bluesky relies on the Ophyd library, that provides a consistent high-level interface across a wide-range of devices. Many elements of the accelerator, some beamlines and endstations are adopting the Bluesky software. To meet FAIR data obligations, the capture of metadata with Bluesky and the export into a permanent and easily accessible storage called ICAT are investigated [3]. Finally, initial studies to investigate the integration of ML methods, like reinforcement learning [4] were performed.This paper reports on the work that has been done so far at BESSY II to adopt Bluesky, problems that have been overcome and lessons learned.

INTRODUCTION

The modernization strategy for experimental DAQ at BESSY II [1] uses EPICS as the unique integration layer for subsystems and software packages. There are many different solutions in use on the experimental floor for the layer above EPICS at BESSY II. This layer has to facilitate experimental flow control, data and metadata collection, storage and analysis.

While various home-grown solutions are in use at BESSY II, most beamlines use spec [5]. This popular and flexible tool has an easy to learn command line interface, has been deployed for decades in production at facilities around the world and is known and understood by many beamline staff. However it is not open source, has a tiny developer community, uses a language that is not well known outside the research community and has no error checking.

Open source alternatives including Sardana [6], PShell [7] and Bluesky [2] were all considered. The Bluesky software ecosystem was judged the most promising tool. Many other synchrotron facilities in Europe (Alba, PETRA III, ESRF, ELI NP, MAX IV, SOLARIS ...) use Sardana interfacing with TANGO. In principle it's possible to use EPICS with Sardana, but the underlying data models of the frameworks differ extremly. Due to the complex adaptation effort required, no facility is doing this in production. Bluesky and Ophyd interface natively with EPICS and there is a large user community in the US (NSLS II, APS, LCLS, SSRL, ALS) and a growing community in Europe (BESSY II, MPG/FHI, Diamond, PSI/SLS ...) and around the world (CLS, ANSTO, PLS II). Like Sardana, it's also based on Python. The data model of Ophyd devices is close to a corresponding TANGO device server instance. That might open opportunities to integrate complex TANGO units into Bluesky or take advantage of Sardana controlling Ophyd devices. Using a tool based on well known language with widely available training has made it easier to bring new people into the project.

This paper will report on progress in the deployment of Bluesky at the facility. First background on the infrastructure of BESSY II and a specific case study, the Energy Materials In situ Lab (EMIL) beamlines, is described. Then component integration, experimental flow control, data and metadata collection, user interfaces, and integration with machine learning (ML) tools are all explored.

SPECIFICS OF BESSY II

From the accelerator commissioning test bed, Bluesky made its way to the experimental floor via instrument integration at the EMIL beamlines. With the aim of being able to collect metadata about the state of the entire beamline when running Bluesky plans, each element of the beamline was given a Python device abstraction using the Ophyd Python package. These general devices classes were then easily transferred to create an interface for a novel beamline project (U49/2 PGM-2, "Aquarius") and to aid replacement efforts at a spec [5] automated beamline (μSpot). Work now continues to integrate other beamlines.

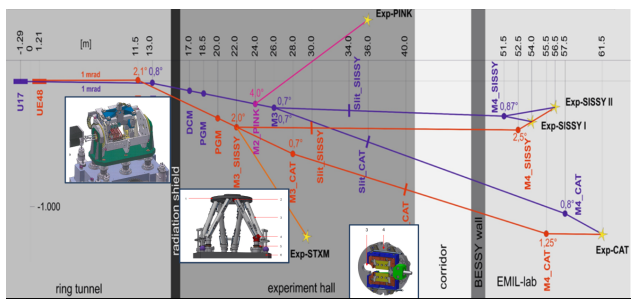

Figure 1: Complex beamline switch yard of the EMIL featuring multiple end stations, instruments and use cases.

CASE STUDY ON EMIL

The initial work to investigate deploying Bluesky at the beamlines was carried out at EMIL. EMIL comes with numerous additional challenges that are due to the complexity of the set-up and the variety of instruments to serve. Plain vanilla EPICS device control of the beamlines comes with EDM based engineering screens, the monochromator controls is based on an integrated VME application, obscur-

* Work funded by BMBF and Land Berlin
† william.smith@helmholtz-berlin.de

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBR03

ing motor controls, beamline commissioning relied on spec macros, continuous energy scan requirements are not eased by custom hardware connectivity, the instruments are owned by the independent legal bodies HZB and MPG. Nevertheless the whole EMIL set-up can be seen as a pathfinder project easing the general roll-out of Bluesky to the inventory of BESSY II beamlines.

COMPONENT INTEGRATION

Ophyd allows us to solve one of the biggest problems with EPICS V3, that records have no inherent structure. EPICS V7 addresses this problem but it's not currently widely used outside areaDetector. Ophyd provides a framework to collect various signals connected to records together into groups which together form devices. It's then possible to read the status of an entire device in one go, and importantly for the collection of metadata it automatically gives context to individual parameters.

It does not solve the problem that values of components of a device can change while they are being read. This can only be solved at a lower level than the IOC to ensure that parameters update synchronously.

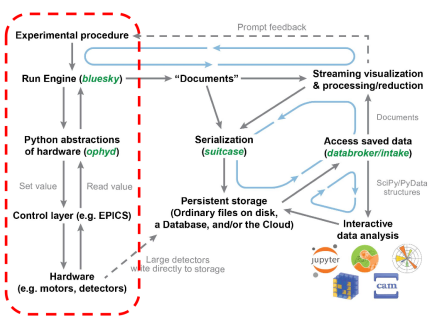

Figure 2: Device integration, abstraction.

Following the example set by NSLS II, LCLS and other labs a set of ophyd devices were created for every element of the EMIL beamlines. This was challenging because of a lack of standardization between different EPICS interfaces for different devices, and even within the same device types at BESSY. None of the motors on the beamline used the EPICS Motor Record, which meant that bespoke device classes had to be made and tested for each type. Despite this challenge the Ophyd framework proved very flexible, all devices were eventually integrated.

Of particular note were the devices on the beamline and at the end station which used the epics areaDetector interface. This included electron analyzers, cameras and spectrometers. Ophyd provided relatively easy integration for all of these, as well as providing the ability to use and control the areaDetector plugins.

Having created Python interfaces for devices, the use of pytest [8] for device level integration was investigated. The existing suite of tests used by the Ophyd package help to guarantee that the abstraction interface itself works as expected. Tests were created to test that physical systems operated correctly. This was particularly important for the positioner device classes that had to be created because the standard EpicsMotor interface was not available. Various bugs that would otherwise have been missed were found by test scripts that attempted to move every motor on the beamline. More work is needed in this area at the beamlines but the ability to perform full integration level tests from Ophyd, through EPICS and to real hardware is promising.

In the accelerator, tools for aperture scans and a power supply multiplexer were developed and tested using a digital twin with an Ophyd interface to simulated hardware. By changing the EPICS PV name prefix passed to the Ophyd device it could be switched to point at the real hardware. A similar approach was used to develop a continuous energy scan plan for the beamlines using Ophyd to connect first to simulated and then real hardware. Digital twins were critical to the development of these tools because there are not spare monochromators or accelerators to test on, and the time given for commissioning is very limited.

EXPERIMENTAL FLOW CONTROL

The commissioning of the EMIL beamlines was performed with spec. The task of converting these spec macros to Bluesky plans was relatively painless. Common spec macros like ascan have similar Bluesky equivalents. More bespoke macros that set up various elements of a beamline could also be easily converted.

Ultimately a beam time manager state machine will be needed, orchestrating sub use cases and handling the variety of experimental plans. This will be based on scripts and semaphores at the point of state transition.

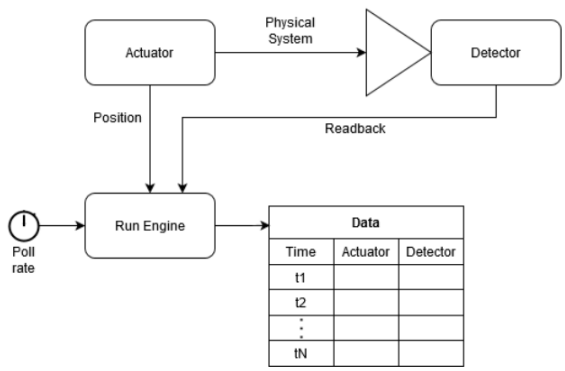

Figure 3: Values are grouped together in continuous energy scans by polling from the Bluesky run engine.

Continuous energy scans, in which a monochromator is commanded to move between certain energies at a certain rate while values are read from detectors on the beamline is implemented in a Bluesky plan. Values from the detector(s) and the energy of the monochromator are polled by the run engine at a fixed rate. The monochromator is assumed to move slowly enough that the error introduced by the energy and detectors being read at slightly different times is insignif-

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBR03

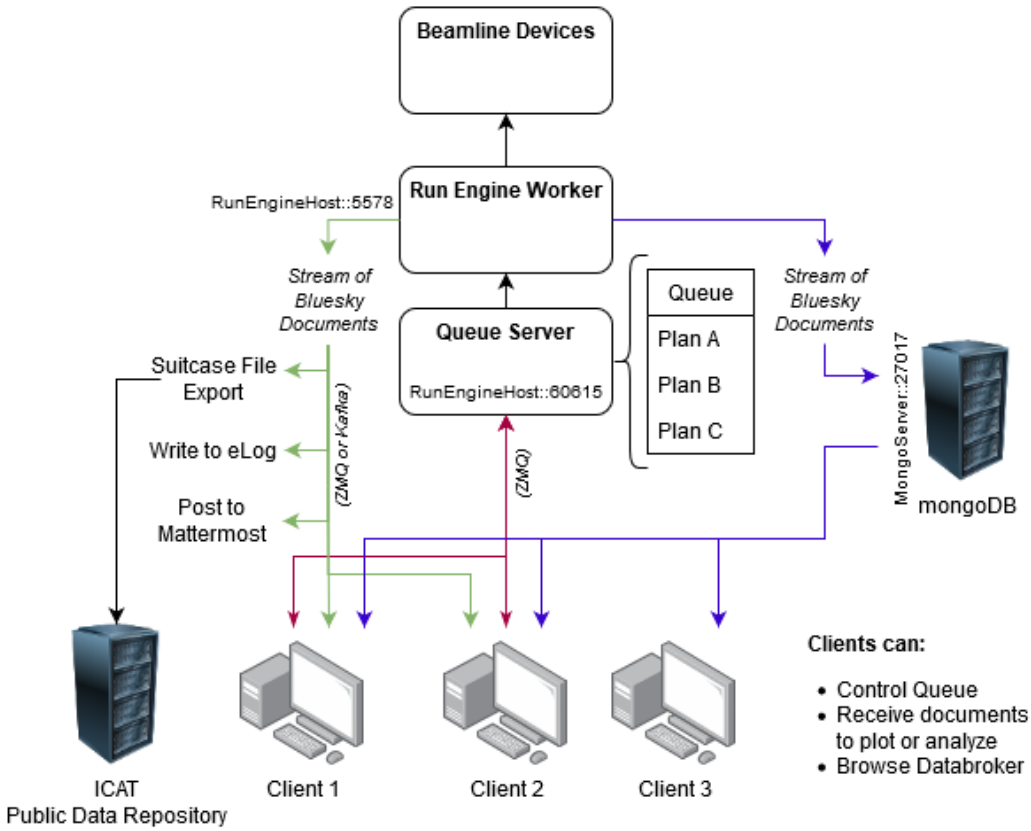

Figure 4: Connections between run engine and clients.

icant. Values of energy and detector readouts are presented live to the user. Bluesky offers a flexible framework for implementing this kind of scan, allowing for the integration of hardware trigger systems to replace polling where necessary in the future.

At both the beamlines and the accelerator the Bluesky run engine is currently interacted with directly, i.e it runs in the same process as the IPython session or Jupyter Notebook that it's run from. This is sufficient for our current use cases, but we are investigating running the run engine as a service using the Bluesky-Queueserver [9] package developed by NSLS-II (See Fig. 4). Operating the run engine as a service means that high importance tasks like data acquisition can be separated completely from less important tasks like displaying live data to clients, or writing to an eLog. If a graphical interface crashes, the run engine will keep performing the experiment.

The Queueserver additionally gives the ability for multiple clients to queue plans, and observe what is currently running. For complex beamlines like EMIL, user access rights for different beamlines or endstations change regularly between shifts. For example, in one shift one endstation might need control of both beamlines, and in the next two different endstations might need to control one beamline each. Using existing access controls based on the EPICS channel access gateway would be one solution, but is seen as being too coarse and is problematic to implement on endstation machines which tend to have common experimental user accounts. It's hoped that adding logic based on agreed scheduling to the Queueserver and user authentication of the clients will allow us to avoid problems we currently experience with multiple simultaneous remote users affecting each other. Users should be allowed to run only particular plans with particular devices during times agreed beforehand. This idea will require considerable development.

DATA AND METADATA

Documents generated by the Bluesky run engine are saved in a mongoDB. For scaler values and most waveforms, data is saved directly in the database. For images produced by detectors with an EPICS areaDetector [10] interface, Bluesky can instruct the EPICS IOC to save images on fast storage with unique names defined by the run engine. Those unique names and file locations are then saved in the metadata of the run stored in the mongoDB. When a client later wants to access the data from a run using the unique run identifier the Databroker [11] package fetches the image from the storage location and presents it to the client. To the client this is transparent, it is as if the image was saved in the mongoDB.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBR03

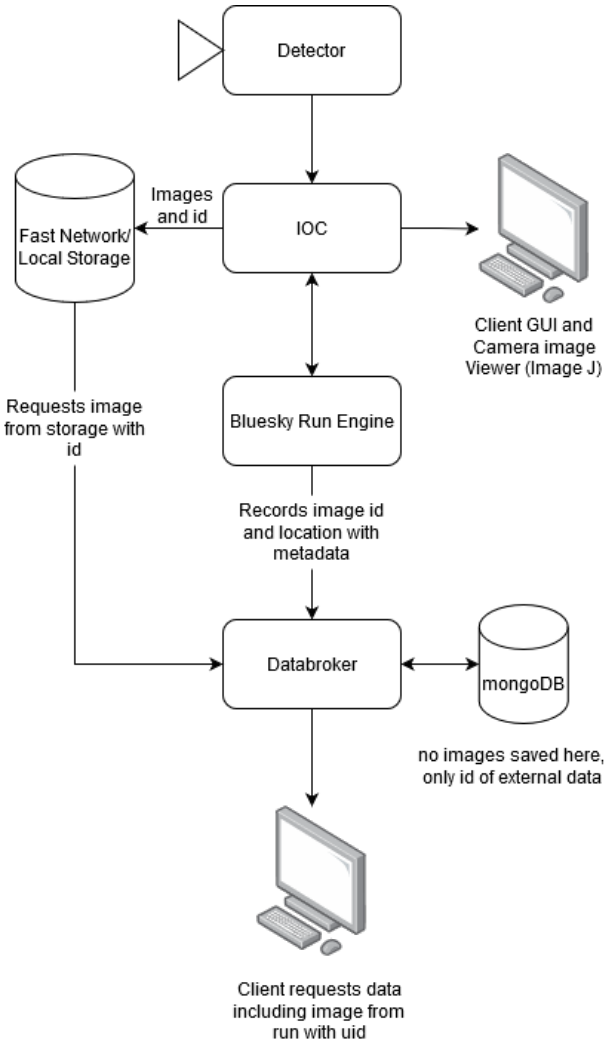

Figure 5: Accessing images by reference with Databroker.

Separating images from the metadata reduces the requirements for the mongoDB database storage and network.

Bluesky is critical to BESSY II achieving it's FAIR data objectives. Every measurement taken with Bluesky can also acquire values describing the state of the entire machine. The structuring of devices with Ophyd gives context to the names of parameters.

Callbacks can be defined which injest documents from the run engine. These callbacks allow metadata to be harvested automatically about samples, repetitive important information added automatically to eLogs, and data and metadata to be exported to publicly accessible repositories like ICAT in standardized Nexus format. This was explored in a case study at EMIL [3]. Notably because documents can be subscribed to over ZMQ, these processes can be separated from the critical task of measuring and saving experimental data.(See Fig. 4)

USER INTERFACES

To allow for simple overview representations synoptic screens will be needed showing positions within their limits, status information, health of devices. For these more engineering screens CS-Studios Phoebus will be used.

Interacting with Bluesky can currently be performed in three ways. Each has benefits and drawbacks, and each is prefered by different users. All are being supported and developed at BESSY II.

IPython is the most mature and popular means of interaction, and the most familiar for users of spec. It can easily and reliably produce live plots of acquired data. Browsing acquired data through the Databroker is more cumbersome than the other interfaces.

Jupyter notebooks can also be used. It has been challenging to get live plotting to work consistently in this environment. For our users who like using Jupyter notebooks, the benefits of the environment have outweighed this issue. The ability to create demonstrations which can be run and shared has been useful for training and collaboration.

The package Bluesky-Widgets provides collection of QT widgets for standard tasks like browsing and plotting from the Databroker, interacting or monitoring the Queueserver, or editing and running plans. These can be integrated with existing QT based GUI's, but at development at BESSY II is focusing on building on the demo application provided with the package. The application is an excellent base which can be adjusted to the needs of a particular user.

Initial deployment of Bluesky at EMIL used the IPython environment, and exported data to .spec files which could then be opened with PyMca [12]. This provided the easiest path to getting a working system that was familiar to beamline scientists.

INTEGRATION WITH MACHINE LEARNING TOOLS

An initial study was performed to investigate using a Reinforcement Learning (RL) agent with the Bluesky Run Engine (RE) with the aim of reducing harmonic orbit perturbations. An inhouse python package, called *Naus*, allowed for communication between the RE and the well established Tensorflow RL agent using ZMQ. Unfortunately this simple demonstrator set-up proved too slow to be used in production. Consequently developments of tweaked sub packages Bluesky-RL, Bluesky-Queueserver and Bluesky-adaptive are being followed now. Hopefully the *Naus* package can be refactored to take advantage of the Queueserver framework.

OUTLOOK

Considerable progress has been made in the last year, and the success of the work at EMIL and at the accelerator bodes well for the roll out to other beamlines. A team of 6 engineers across different departments is now working together to develop the BESSY II Bluesky ecosystem. Bluesky implementation and development moved to the core of a BESSY II reinforcement and modernization program. Visible achievements are convincing enough to ask for additional support of the funding agency.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-FRBR03

A route to making data acquired at the beamlines FAIR has been identified, and Bluesky has made it possible. Further work is needed to put this into production, not least making it easy for users to authenticate and associate their session with an investigation ID.

Authentication will also play a role in the development of access control at complex beamlines with shared resources like EMIL. Further work is required to understand how this can be incorporated with Bluesky.

The pathfinder study looking at integrating Bluesky with RL agents showed promise, but was not fast enough to be used in production in the accelerator. The work at other facilities on this topic is being followed closely.

The Bluesky user community continues to grow at labs of all sizes around the world. The energy and solution focused attitude of the people involved is one of the projects biggest assets.

ACKNOWLEDGEMENTS

The authors thank all HZB staff members active in maintaining the complex controls infrastructure at BESSY II for many fruitful and clarifying discussions. This includes beamline and instrument scientists, accelerator controls, beamline optics as well as central IT networking and storage.

Additionally we would like to thank the developers of the Bluesky ecosystem for their work and help with our many questions.

REFERENCES

[1] R. Müller, A.F. Balzer, P. Baumgärtel, G. Hartmann, O.-P. Sauer, and J. Viefhaus, "Modernization of Experimental Data Taking at BESSY II", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 65–69. doi:10.18429/JACoW-ICALEPCS2019-MOCPL02

[2] Bluesky Project, https://Blueskyproject.io/.

[3] G. Gerrit, "FAIR Meets EMIL: Principles in Practice", presented at ICALEPCS21, Shanghai, China, Oct, 2021, paper WEBL05, this conference.

[4] L. Vera Ramirez, "Machine Learning Tools Improve BESSY II Operation", presented at ICALEPCS21, Shanghai, China, Oct, 2021, paper THAL01, this conference.

[5] spec, Certified Scientific Software, https://certif.com

[6] https://Sardana-controls.org

[7] Pshell,Alexandre Gobbo, PSI, https://github.com/paulscherrerinstitute/pshell

[8] pytest is a framework that makes building simple and scalable tests easy https://docs.pytest.org/en/6.2.x/.

[9] server for queueing Bluesky plans, https://github.com/Bluesky/Bluesky-queueserver

[10] A tool for controlling 2D detectors, https://areadetector.github.io/master/index.html

[11] A data access tool for Bluesky, https://Blueskyproject.io/databroker/.

[12] X-ray fluoresence data analysis, http://pymca.sourceforge.net/.