ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

PYTHON FOR USER INTERFACES AT SIRIUS

G. S. Fedel†, D. B. Beniz‡, L. P. Carmo§, J. R. Piton#
Laboratório Nacional de Luz Síncrotron, Campinas, Brazil

Abstract

Sirius is the new Brazilian Synchrotron and will be finished in 2018. Based on experiences at LNLS UVX light source along with researches and implementations, we present our new approach to develop user interfaces for beamlines control. On this process, the main tools explored are Python, Qt and some Python libraries: PyQt, PyDM and Py4syn. Powerful resources of these modules and Python straightforward coding guarantee flexible user interfaces: it is possible to combine graphical applications with intelligent control procedures. At UVX, EPICS and Python are software tools already used respectively for distributed control system and control routines. These routines often use Py4Syn, a library which provides high-level abstraction for devices manipulation. All these features will continue at Sirius. More recently PyQt turned out to be a compatible and intuitive tool to build GUI applications, binding Qt to Python. Also PyDM offers a practical framework to expose EPICS variables to PyQt. The result is a set of graphical and control libraries to support new user interfaces for Sirius beamlines.

INTRODUCTION

The Brazilian Synchrotron Light Laboratory (LNLS) started in 1997, building UVX, the first synchrotron light source of South Hemisphere. Today, UVX has 16 beamlines opened for user community. Since its foundation, control software used on UVX beamlines evolved in many ways. Starting with in-house software, EPICS [1] was adopted in 2011 and at the moment our control software solution is mostly based on it. EPICS led to a standard method of accessing different devices. However, other needs emerged at the beamlines regarding data acquisition, such as user-friendly monitoring and control as well as integration with complex experiment routines. In order to address these requirements, LNLS started to research more efficient user interfaces.

At the beginning control procedures at UVX were mostly based on direct access to EPICS PVs (process variables), using commands such as caget and caput. Then, many different ways of providing simple but powerful user interface were experimented. Chosen tools include frameworks written in C, Java and Python, and currently the most used ones are CS-Studio [2] and Python scripts that include Py4Syn [3, 4], library for high-level operations at synchrotrons.

† gabriel.fedel@lnls.br
‡ douglas.beniz@lnls.br
§ lais.carmo@lnls.br
# james@lnls.br

All this experience resulted in good guidelines for what will be selected for Sirius, the new Brazilian synchrotron LNLS is currently working on.

Sirius promises to be one of the most brilliant synchrotron light sources, planned to achieve higher energy and much lower emittance than UVX, with initially 13 beamlines and 40 ones as the final objective. This new synchrotron facility will allow experiments that are not possible today at the current Brazilian light source. The first electron beam is planned for 2018 and the early experiments at beamlines are expected in the course of 2019.

Constructing a new laboratory brings a lot of innovation challenges to all fields related to its project and on control user interface it is not different. Each beamline at Sirius will have different types of devices and they will be more numerous than they are at UVX. Also, different types of experiments will be available per beamline. For robust control of all equipment and experiments, we pursued a framework for intuitive and flexible user interfaces, not only for the end user but also regarding development.

On this process, experiences at UVX in the last years are highly valuable while defining what will be prepared for Sirius. Based on them, we summarize challenges and proposed solutions for user interfaces at Sirius.

CONTROL INTERFACES AT UVX

Providing a control interface for a beamline is not a simple task. Normally, there are many different equipment from different manufacturers and with particular software.

At UVX, since we started using EPICS this problem became simpler. EPICS brings a common interface to communicate with several devices, organizing and accessing their properties by creating process variables (PVs) through an IOC (Input/Output Controller). But EPICS doesn't provide a graphical user interface, but something closer to a middleware interface. Besides that, we chose to let on IOC just low-level code, that is, we didn't insert in IOC complex operation like a motor scan.

On LNLS user interfaces is grouped in two main groups: Simple Read/Write PVs and Experiments. Simple Read/Write PVs are usually used for tasks such as moving a motor, updating parameters of a picoamperimeter, monitoring values from a detector. All these tasks are resumed to read and write values from/to process variables. Experiments are tasks a bit more complex and involve working with different hardware together with performing a set of actions related to PVs. An example is a motor scan, a task where a motor is moved while a detector is read. On motor scan, a motor is moved only if detector has finished acquisition. These experiments are built as Python scripts and could be

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

controlled and monitored by command line or by using a graphical user interface.

Today, at UVX, different technologies are employed to provide user interface and control. Some of them are the C framework MEDM [5] and C-like SPEC [6] macros. There are also tools based on Java such as Labweb [7] and CS-Studio. Python is used as well for command-line scripts that include Py4Syn, or for building GUI applications with libraries suhc as Tkinter [8]. But among all these technologies the most used are both CS-Studio and command line. They are discussed in next session.

CS Studio

The most used graphical user interface tool today at UVX is Control System Studio (CS Studio). This framework is built with JAVA and provides tools for monitoring and controlling large-scale systems, being compatible with EPICS process variables. Operation interfaces (OPI) can be composed with its graphical tool called BOY (Best OPI Yet).

Some advantages of CS Studio are:

- **Open source**: This allows modifications on CS Studio, adapting it for different uses;
- **Integration with EPICS**: CS Studio has native integration with EPICS;
- **Simple installation**: The installation process is very simple, with plenty of ready-to-use resources;
- **Good usability**: Like in Eclipse IDE, in CS Studio graphical interfaces can be built by dragging-and-dropping widgets;
- **Scripts insertion**: Besides widgets, it is possible to insert scripts (in Python or JavaScript) for customize an object behavior. Objects can also be controlled by other supplied mechanisms, which are the configuring of actions and rules.

In spite of these advantages, CS Studio has some limitations. Some of them occur because of features of our development environment:

- **Integration with Python**: Python is used on different levels and sometimes CS Studio integration is very limited for complex operations. Also Python scripts inserted into CS Studio must rely only on libraries available from Jython interpreter (Python implementation integrated with Java). Not all Python modules are accessible through Jython and calling system-wide Python libraries requires additional implementation.
- **Complex scripts degraded performance at CS Studio**: CS Studio is more fitted for tasks related with direct PV reading and writing. The addition of complex scripts in CS Studio consumes a lot of machine resources;
- **Hard to Modify Source Code**: Despite CS Studio being open source, changing its source code is not simple. In many cases a lot of workarounds has to be done to achieve a complex behavior.
- **Control logic and visualization are together**: In CS Studio personalized control can be done though actions, rules and scripts, which may be applied to a given object. Although these features are user-friendly, they can make the resulting application difficult to understand and bugs hard to track, once the control logic becomes fragmented and merged to the graphical portion.
- **Too much resources** High demands of RAM and CPU resources are faced in CS Studio at UVX, specially when it comes to operation interfaces containing numerous process variables being monitored and/or embedded Python routines. This often leads in undesired situations like frozen screens, damaging user experience.

These limitations was the motivation to find alternatives for graphical user interfaces that could better match our environment, considering all peculiarities and technologies within it.

Command Line

Command line are still used at UVX due to two major reasons: legacy from other technologies and possibility to construct more complex experiments. The first reason comes from scientists experience with other technologies like SPEC, that are completely controlled by command line. The second one is the facility that command line permits while combining commands in a simple way and building more complex experiments. Command line is broadly used by other synchrotrons and this contributes to legacy use on UVX.

At UVX command line programs are built by using Python and some libraries, e.g, PyEpics [9]. To simplify and standardize development process some new libraries were build, e.g., Py4Syn, which combines a lot of common methods used in synchrotron experiments. Py4Syn provides mnemonic commands that are intuitive or even familiar to the users, commands like scan, tscan and others.

These scripts could be used on command line, but sometimes they are integrated into a graphical interface built with CS Studio.

Alternatives

Limitations found while using CS Studio wast the motivation to look for alternatives tools for graphical user interfaces. Moreover, more efficient ways of combining user interfaces with control procedures were searched. The main objective was to find tools that better fit our environment, resolving current challenges and preserving features that are already good.

Challenges that needed to be solved are better integrating graphical user interface with Python routines, separating control logic and graphic components for more organization and code reuse,and better management of computational resources. Features to preserve are user interfaces integrated to EPICS, with easy installation and intuitive usability.

Another key feature is maintaining both command line and graphical interface as means of user input. However, it is important not only to maintain them available, but improve their communication with control logic. A good scenario

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

would be having a control routine possible to be reused, by being called by either command line or graphical interface.

When it comes to alternatives for graphical user interfaces, different GUI software were tried, mainly inspecting those currently used by other synchrotron laboratories. Interesting solutions where found in tools that combine Python and Qt [10] framework, such as PyQt [11]. At the beginning, very simple screens for monitoring and controlling PVs were created and latter a few simple beamline GUIs implemented with CS Studio were selected and remade with PyQt. On this process, the need of a framework for linking GUI to channel access emerged and, at this moment, PyDM [12, 13] library turned out to be very helpful. PyQt and PyDM are detailed in the following section.

Regarding communication between user interfaces and control logic, experiments with parallelism were made, exploring thread-based and process-based approaches. On the last one, the most interesting option was D-Bus, a well-established protocol in Unix and Linux environment for Inter-process Communication (IPC), which is better explained in the next section.

SELECTED TOOLS

Building a new approach for user interfaces requires a set of tools that allows functionality similar to already tools in use (such as CS-Studio). Almost all chosen tools are based on Python and under Open Source license.

The combination of these tools on synchrotron is a contribution of this work.

PyQt

PyQt is a Python library which provides bindings for C++ Qt cross-plataform application framework. It is currently developed by Riverbanking Computing and works with both Python version 2 and 3, supporting Qt version 4 (PyQt4) and 5 (PyQt5). PyQt is available under GNU GPL version 3 and Riverbank commercial license for all supported platforms.

Qt framework is widely employed in application development for desktop environment. It is mainly used as a GUI tool, known by its rich collection of graphical objects (widgets). However it also offers other resources as threads, network sockets and D-Bus [14] implementation. One of the main features of Qt is the signal/slot mechanism which allows communication between objects, making application development very intuitive. Qt also offers Qt Designer, a graphical environment for creating graphical applications without coding, by simply drag-and-dropping widgets. Customized widgets and controls can be easily created by deriving Qt classes and extensions can be loaded into Designer by its plugin mechanism. User interface (UI) files created inside Designer are described by XML code which can be latter translated to other programming languages.

The main advantage of PyQt is combining the powerful resources from Qt framework with the simplicity of Python interpreted programming language. The tool used for binding Qt to Python is SIP [15] which encapsulates C++ Qt libraries. PyQt in organized in Python modules and makes available more than one thousand classes. A Python plugin is included so extensions for Qt Designer can be written in Python. PyQt can convert Designer UI files to Python, making them possible to be integrated to Python applications. PyQt, as well as Qt and Python, has currently a large documentation and community support.

Python and Qt are flexible frameworks that fit in general purpose. They find place in a wide range of contexts, both meeting frequent application in scientific experiments for control and monitoring [16–18]. PyQt already can be found at several synchrotron laboratories [16, 19, 20]. Examples of applications based on PyQt currently in use at LNLS and in other synchrotron facilities also include PyMca[1] [21] , Orange[2] [22] , MXCube[3] [23, 24] and PyQtGraph[4] [25] .

PyDM

PyDM is a practical framework based on PyQt for building graphical user interfaces for control systems. Its development started at SLAC. PyDM combines PyQt resources to build a simple way to connect graphical objects to a channel access. Different channel access are admitted, e.g, EPICS which is reached by PyEpics library. A set of widgets is included and can be used as Designer plugins for easily designing screens. Also a graphical application is available for quickly loading UI files. In addition, its simple API makes possible to build extensions and complex applications.

QtDBus

D-Bus is a system bus which allows applications to communicate by messages on UNIX and Linux. It is not only useful for inter-process communication, but also for managing process lifecycle such as launching an application or daemon and terminating them when needed. High level bindings for D-Bus are found written in various programming languages such as C#, Java and Python. Qt has its own implementation of D-Bus, the module QtDBus, which is also part of PyQt.

For using D-Bus, a process has to connect to a message bus daemon. A bus daemon can be seen as a router, forwarding messages from an application to another. Once connected to the bus daemon, a process may expose its objects, making possible to other process to access members of an object, such as methods and signals. D-Bus process communication through messages also applies for the case when applications are running on different machines.

QtDBus available in PyQt provides a very intuitive interface to work with D-Bus resources. It also extends the signals and slots mechanism, then a local signal can be connected to a remote slot as well as a local slot can be triggered by a signal remotely emitted.

[1] X-ray Fluorescence toolkit
[2] Machine learning and data visualization toolbox
[3] Environment for macromolecular crystallography beamlines
[4] Pure-python GUI library for fast display in scientific applications

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

CONTROL INTERFACES AT SIRIUS

Considering experience with user interfaces at UVX and studies about alternatives, a new, standardized method to build user interfaces for Sirius was developed. For this new approach, technologies were selected to form the basis of a framework.

Two different types of user interfaces are still adopted: graphical and command line. The command line interface will not have great changes on how user interacts with it. Back-end code, however, was modified in a way that its structure can be called by graphical interface or command line.

A Framework for User Interfaces at Sirius

For Sirius a framework for user interfaces was defined, which introduces how user interface is implemented, as well as how it is connected to lower-level elements. The framework can be applied in two different ways as presented in Figures 1 and 2.

In Figure 1 the simplest case, in which interface it is connected directly to EPICS by using PyDM library, is show. In this case, it is possible to build user interfaces by using just QtDesigner and PyDM. An example of such user interface is one for simple motor control as shown in Figure 3.

In Figure 2 the experiment case is presented. This scenario covers experiments such as motor scans. An experiment server contains experiment control logic and it can be accessed either by graphical interface or command line. In this case a client/server model is used, where user interface is a client for experiment server, while the latter communicates with EPICS. The experiment server reuses a lot of pre-existing code from UVX repository.

Pre-existing code, which was mostly written sequentially, was re-factored to match object orientation, being reorganized into classes. This way, that code became easier to be incorporated within our new approach. At experiment server third-party libraries are included, PyEpics and Py4syn, for instance.

An important feature of this framework also shown in Figure 2 is the communication between user interface and experiment server. These applications exchange messages through QtDBus. Thus, the experiment can notify user interface about its current state. Then user interface can be updated accordingly so the end user can be aware of experiment course. Likewise, user interface may send commands to the server, triggering experiment tasks.

A relevant aspect of this framework is that most of its pieces could be replaced by other implementations with low effort. For example, it is not hard to replace PyQt by another GUI library (e.g. PySide or Tkinter). This makes our framework flexible and not bound to a specific technology.

In this paper lower-level framework (from EPICS to equipment) is not presented, but this is described in details on [26].

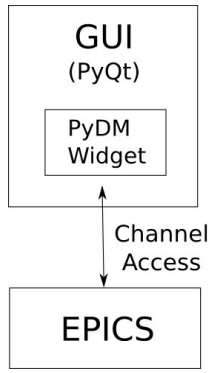

Figure 1: Framework for direct access to EPICS PVs.

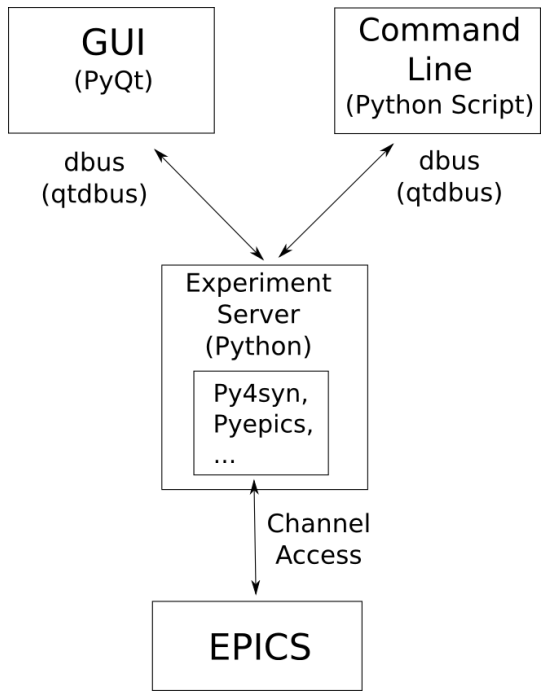

Figure 2: Framework for Experiments.

Screen Previews

Here a few screenshots of GUIs are shown, some in use and others for future use. These screens come from applications at beamlines and have been developed by LNLS software control group. The technologies presented on this work (like PyDM and PyQt) are being used in other areas for Sirius such as the accelerator physics group (for monitoring accelerators) and the power electronics group (for equipment tests).

In Figure 3 a simple motor control interface is presented. This interface is assembled with PyQt with PyDM plugins, connecting it to motor PVs, which follows the model shown in Figure 1. As motor operation is a very frequent task, this interface is thought to be a plugin widget for Qt Designer, in a way it can be easily reused by various screens.

In Figure 4 a scan motors interface is shown. This interface is used to perform a scan using an arbitrary number of motors and counters (e.g. scalers). This interface fits in

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

Figure 3: Simple motor interface.

the framework structure described by Figure 2, given that there is an experiment server which performs scan based on parameters set on the the interface. The Counter Set option allows users to retrieve sets of counters for configuring a specific experiment.

In Figure 5 an interface for Ocean Optics Spectrometers is presented, using David Beauregard's IOC [27]. This interface is a hybrid case between framework on Figure 1 and 2. Nearly all of the interface was built using PyDM widgets, while the right lower area was customized for a simple "time scan", performing an acquisition series and saving the result. This part was implemented by using an experiment server and simple PyQt components.

Figure 6 shows a interface for baking procedures. This interface was built using PyDM widgets only, with routines added just for simple customization of graphical behavior. The controls on this screen allow that temperature curves are set by user. Also multiple temperatures sensors can be monitored in a single time plotting widget from PyDM. This widget is derived from PyQtGraph library and some of its features, such as the possibility of drawing multiple curves together, were implemented with LNLS participation.

Next Steps

The development of user interfaces for Sirius is still in progress, thus some topics are open and will be defined soon. Some of them are listed below:

- **Standardize common windows**: Many devices and processes are common at almost all beamlines. Some examples of usual operations are motor control and configuration, slit control and motor scans. For these cases, standard interfaces have been developed to be available for all beamlines. Besides common functionality, it is intended to set design guidelines, defining style parameters such as color schemes and fonts. This features will give the beamline scientists a more comfortable user experience since all of them will have their GUI with similar operation and appearance.
- **Access Control**: Implementing access control in user interfaces, providing different views for each level access (e.g. administrator, final user).
- **Web access**: The proposed framework probably could be used for web access without extensive changes. In this case, it would be necessary to define and implement other layers for web interface. Web user interface was experienced with Labweb, but all the code come from a different resource, what generates duplicate effort for maintenance.

CONCLUSION

On this work a framework for user interfaces was described, based mostly in Python, which led to fast developing and deployment and easy integrating to other parts of a beamline system. The result was functional controls with simplicity at the same time. Code reusing and open-source solutions were also important aspects on this process. The initial results using this model, with PyQt and PyDM, are very promising and are already they are being used at some beamlines on UVX. All the work presented here will be used at Sirius, the new Brazilian synchrotron light source.

REFERENCES

[1] EPICS, http://www.aps.anl.gov/epics/.

[2] C. Studio, http://controlsystemstudio.org/.

[3] H. Slepicka, H. Canova, D. Beniz, and J. Piton, "Py4syn: Python for synchrotrons," *Journal of synchrotron radiation*, vol. 22, no. 5, pp. 1182–1189, 2015.

[4] Py4Syn, http://py4syn.lnls.br/.

[5] MEDM, http://www.aps.anl.gov/epics/extensions/medm/.

[6] SPEC, https://certif.com/spec.html.

[7] H. Slepicka, M. Barbosa, D. Omitto, R. Bongers, M. Cardoso, J. Polli, D. De, J. Oliveira, C. Rodella, H. Canova *et al.*, "Labweb–lnls beamlines remote operation system," *TUPPC037, Proc. ICALEPCS*, p. 638, 2013. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/tuppc037.pdf

[8] D. B. Beniz and A. M. Espindola, "Using tkinter of python to create graphical user interface (gui) for scripts in lnls," *Proceedings of PCAPAC, Campinas, SP, BRAZIL*, p. 1, 2016. [Online]. Available: http://vrws.de/pcapac2016/papers/wepoprpo25.pdf

[9] PyEpics, http://cars9.uchicago.edu/software/python/pyepics3/.

[10] Qt, https://www.qt.io/.

[11] PyQt, https://riverbankcomputing.com/software/pyqt/intro.

[12] PyDM, https://github.com/slaclab/pydm/.

[13] T. Rendahl, "Pydm: a python alternative to edm," 2016, ePICS Collaboration Meeting. [Online]. Available: http://conference.sns.gov/event/11/session/1/contribution/45/attachments/131/345/PYDM_.pdf

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

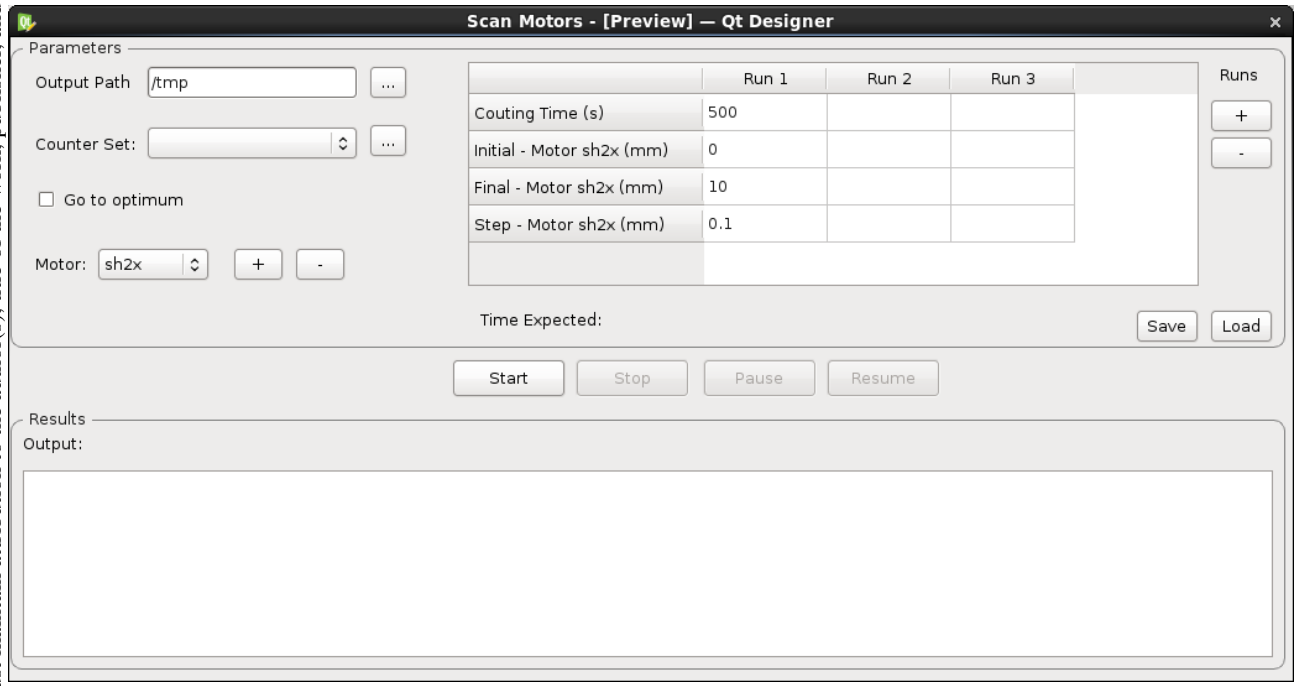

Figure 4: Scan Motors Interface.

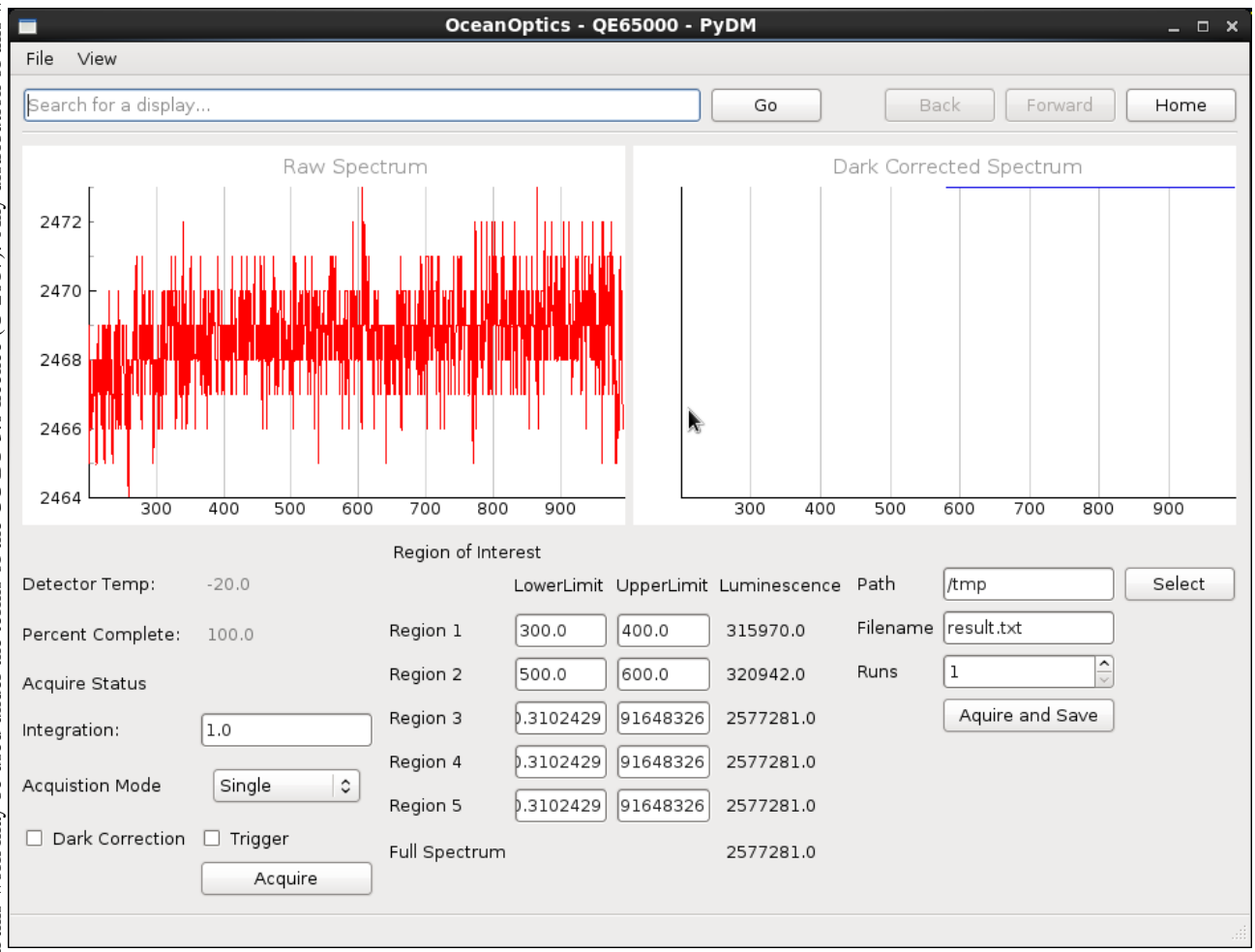

Figure 5: Ocean Optics Spectrometer Interface.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL04

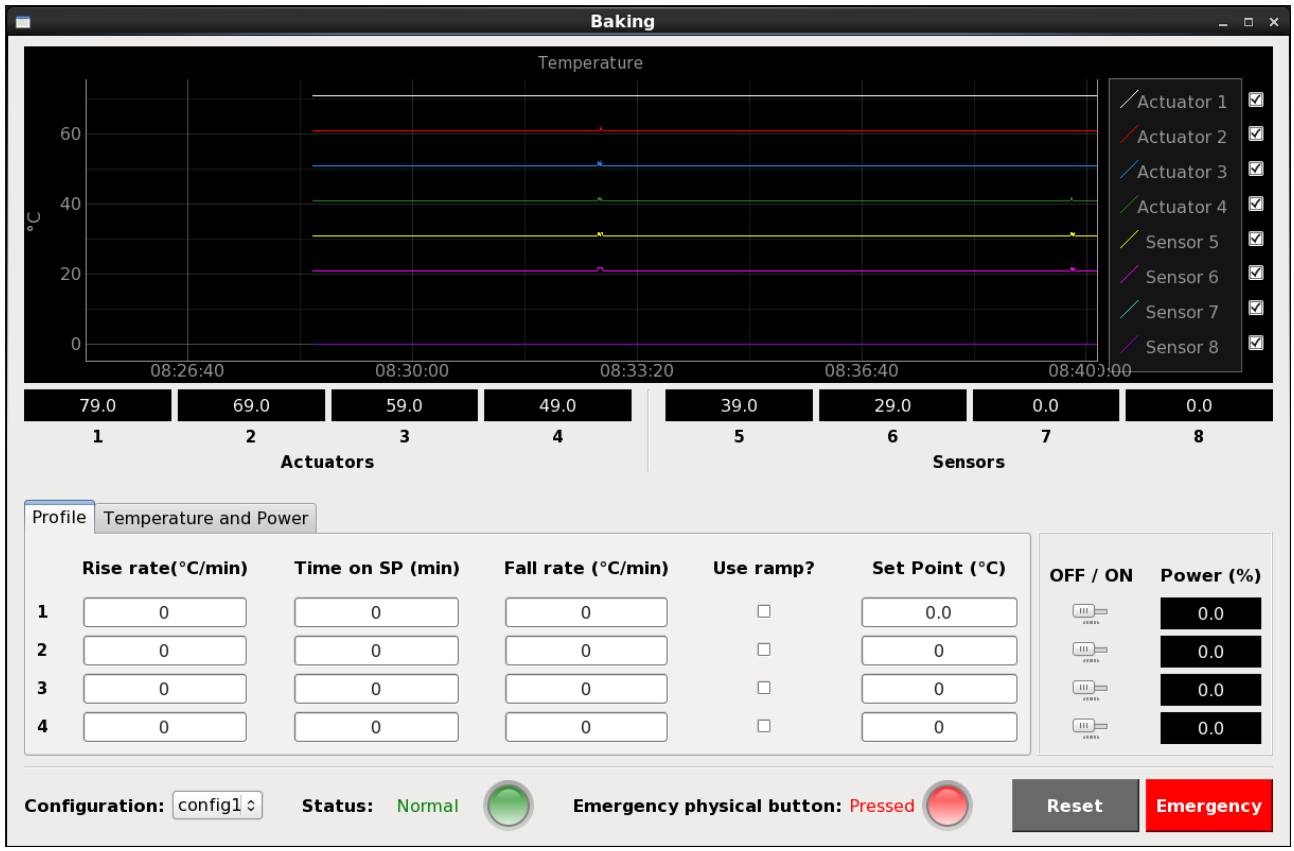

Figure 6: Baking Interface.

[14] D-Bus, https://www.freedesktop.org/wiki/Software/dbus/.

[15] SIP-Documentation, http://pyqt.sourceforge.net/Docs/sip4/.

[16] M. G. Abbott, T. M. Cobb, I. J. Gillingham, and M. T. Heron, "Diverse uses of python at diamond," *Proceedings of PCaPAC08, Ljubljana, Slovenia*, pp. 137–139, 2008. [Online]. Available: http://accelconf.web.cern.ch/accelconf/pc08/papers/tup024.pdf

[17] R. E. Mayssat, "Collaborative development of the epicsqt framework," Lyncean Technologies, Inc., WU Vienna University of Economics and Business, Vienna, Tech. Rep. DE-SC0011283, Novembro 2014. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/1212894/

[18] A. Mezger and H. Brands, "Caqtdm: an epics display manager based on qt," *Proceedings of ICALEPCS2013, San Francisco, CA, USA*, pp. 864–866, 2013. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/tuppc121.pdf

[19] J. Lidón-Simón, V. Hardion, A. Persson, M. Lindberg, A. M. Otero, and D. S. Jerzy Jamroz, "Status of the max iv laboratory control system," *Proceedings of ICALEPCS2013, San Francisco, CA, USA*, pp. 366–369, 2013. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/moppc109.pdf

[20] C. Pascual-Izarra, G. Cuní, C. Falcón-Torres, D. Fernández-Carreiras, Z. Reszela, and M. Rosanes, "Effortless creation of control & data acquisition graphical user interface with taurus," *Proceedings of ICALEPCS2015, Melbourne, Australia*, pp. 1–4, 2015. [Online]. Available: http://icalepcs.synchrotron.org.au/papers/thhc3o03.pdf

[21] PyMca, http://pymca.sourceforge.net/.

[22] Orange, https://orange.biolab.si/.

[23] MXCube, http://mxcube.github.io/mxcube/.

[24] D. B. Beniz, "Customization of mxcube 2 (qt4) using epics for a brazilian synchrotron beamline," presented at the 16th International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS2017), Barcelona, Spain, Oct. 2017, paper THPHA201, this conference.

[25] PyQtGraph, http://www.pyqtgraph.org/.

[26] M. A. L. Moraes, H. D. Almeida, R. M. Caliari, R. R. Geraldes, G. B. Z. L. Moreno, J. R. Piton, and L. Sanfelici, "A control architecture proposal for sirius beamlines," presented at the 16th International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS2017), Barcelona, Spain, Oct. 2017, paper THPHA215, this conference.

[27] D. Beauregard. Ocean optics spectrometer ioc. http://exshare.lightsource.ca/OpenSource/Software/Forms/AllItems.aspx.