ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA201

CUSTOMIZATION OF MXCuBE 2 (Qt4) USING EPICS FOR A BRAZILIAN SYNCHROTRON BEAMLINE

D.B. Beniz, Brazilian Synchrotron Light Laboratory, Campinas, Brazil

Abstract

After studying some alternatives for macromolecular crystallography beamlines experiment control and had considered the effort to create an in-house made solution, LNLS decided to adopt MXCuBE [1]. Such decision was made considering main technologies used to develop it, based on Python, which is being largely used in our laboratory, its basic support to EPICS (Experimental Physics and Industrial Control System), the control system adopted for the LNLS beamlines, and because of its stability. Then, existing MXCuBE implementation has been adapted to cover LNLS requirements, considering that previously it was mainly ready to control systems other than EPICS. Using basic MXCuBE engines, new classes were created on devices abstraction layer, which communicates to EPICS IOCs (Input/Output Controllers), like AreaDetectors, MotorRecords among others. Py4Syn [2] was employed at this abstraction layer, as well. New GUI components were developed and some enhancements were implemented. Now, MXCuBE has been used on LNLS MX2 beamline since the end of 2016 with positive feedback from researchers. The adoption of MXCube proved to be right, given its flexibility, performance and the obtained results.

MOTIVATION

LNLS macromolecular crystallography beamline, MX2, has been reformed to be the base for its correspondent on Sirius, new Brazilian Synchrotron Light Source. During it, software control systems were also revised. EPICS was kept as the basic control system for devices operation, and using that we looked for a better GUI to offer a good experience for researches when performing experiments on MX2. In the past, some GUI solutions were tested on such beamline, like Blu-Ice [3] and an in-house development based on CS-Studio [4], but they were not totally well-succeeded.

Because MX2 coordinator, Ana C. M. Zeri‡, had experienced the usage of MXCuBE on ESRF, she asked software support group of LNLS to take a look at it as an alternative, and then we started to customize it for our environment.

ARCHITECTURE OVERVIEW

Main organization of MXCuBE Qt4 was kept untouched; the basic support for EPICS was used but modified in some parts to allow the usage of Py4Syn. As other laboratories that contribute to MXCuBE development do, we created new folders named LNLS on layers that abstract hardware objects and their configuration. This way, we could develop our own Python classes with procedures performing operations according to our equipment and necessities.

An overview of MXCuBE architecture we customized to offer full experiment operations control for researchers of MX2 beamline at LNLS is presented on Figure 1.

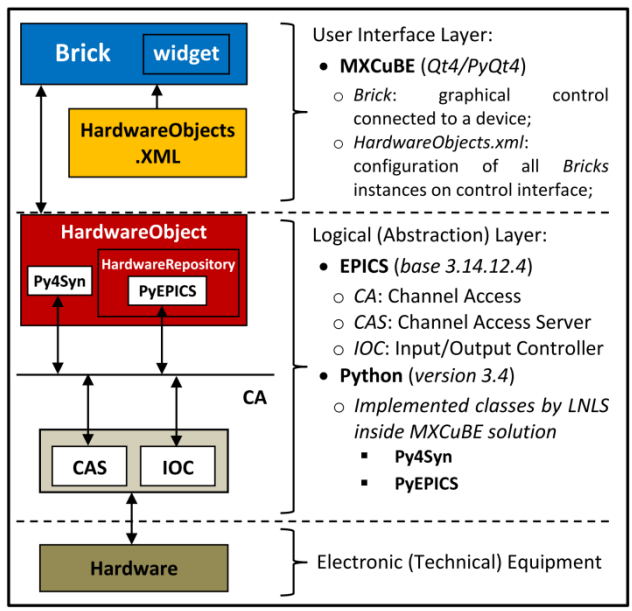

Figure 1: Overview of MXCuBE for LNLS Solution Architecture

Electronic (Technical) Equipment

At lower level of control system are the typical technical devices present in synchrotron beamlines. In fact, they have their own controllers which receive instructions, via serial (RS232/RS248) or Ethernet connection, for example, and then command the equipment. Some devices present in MX2 beamline of LNLS are:

- Galil DMC-4183: motor controller
- Parker OEM750: motor controller
- Kollmorgen S300: motor controller (air bearing)
- Heidenhain MT 2501: optical encoder
- Keithley 6485: picoammeter
- Cryojet: temperature controller of cryogenic cooling system
- Stanford SR570: low noise current preamplifier
- Dectris Pilatus 2M: CCD camera
- IDS GigE uEye: industrial camera (view sample)
- Stäubli CS8: robot controller (sample changer)

Logical (Abstraction) Layer

Over the devices controllers is the first abstraction of them, build in EPICS, with correspondent IOCs

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA201

(*Input/Output Controller*) for each one of the devices. Those devices of the same manufacturer and model share the same IOC program, but run in individual instances. Basically, a set of instructions to get information or to send a command to devices is organized in those IOCs as PVs (*Process Variable*), where each PV is a record, or piece of data, with some attributes to format, configure or simply return related information.

All PVs are broadcasted in the network via CA protocol, in which subnet where CAS is connected they are accessible.

The second abstraction level of those PVs is made by Py4Syn, which offers a set of Python objects representing each one of devices controlled by EPICS IOCs. Py4Syn also implement a set of utility tools to perform scan of motor positions while a counting detector is accumulating information of beam intensity transmitted across a photodiode, or to vary a goniometer angle while a CCD is acquiring spectra images to analyse the x-ray diffraction pattern produced by a crystal sample, for example. It also allows a combination of motor movements, like a mesh of two motors, and mathematic calculations based on the measure of one or more detectors.

User Interface Layer

Finally, in the top layer of this architecture is the GUI. The focus of this article is the adaptation of MXCuBE, a graphical interface very stable and used by important macromolecular crystallography beamlines in different synchrotron light sources on Europe, e.g.

The version we adopted is based on PyQt4, which is a library that encapsulate Qt4, via SIP, that allows C/C++ programs to be called inside Python scripts. Qt4 has an easy framework to create graphical interfaces, offering some widgets of components that facilitate interaction between users and the operation the programs execute, like input text boxes, labels, command buttons, tables, graphics, images, etc. An utility of Qt4, called Qt Designer, is used to produce an XML file with all necessary information to generate a GUI, by simply dragging and dropping widgets and configuring their characteristics, like name, to be used by Python scripts to bind procedures to them, text to display, size, position, between others.

MXCuBE has a design operation mode which generates an interface with all components that, together, allow the researchers to perform experiments on a MX beamline, like that in Figure *2*.

Since all the components are in desired places, it is necessary to program what is called “bricks” in MXCuBE. Each brick is a basic graphic control element, and could combine some components, from where information and actions are received from users, and to where processing results, alerts and error messages are displayed. To produce the final interface, a set of bricks are configured to control physical devices, so, some instances of such programed bricks are running on a final interface during execution mode. For example, a brick of a motor contains the graphical components to receive desired target position and to start or stop the movement, and the logic binding it to a hardware object abstraction, which is responsible to send commands to real motor. To instantiate a set of motors it is just necessary to configure them on an XML file. PyQt4 has a concept of signal-slot communication between the graphical and logical layers, and using it MXCuBE send information from bricks to hardware objects.

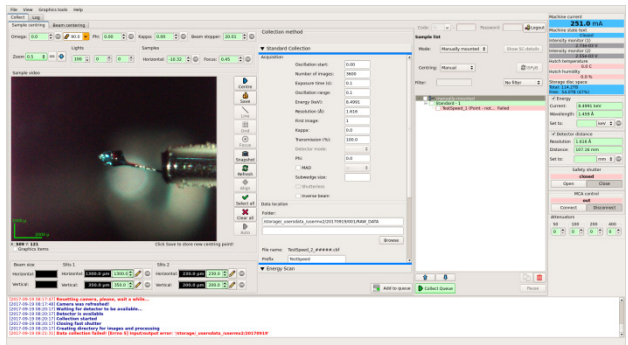

Figure 2: MXCuBE main interface for MX2 beamline.

MXCuBE SOLUTION FOR MX2

It was necessary to create some EPICS IOCs, like one for IDS uEye industrial camera, following *AreaDetector* standards and using AravisGigE [5] driver. The result, after encapsulate the treatment of image returned as an array by such developed IOC in a Python class inside hardware object layer of MXCuBE, is that showed in Figure 2.

To bind such EPICS IOCs with Python scripts, inside MXCuBE, we also used Py4Syn, inside HardwareObject layer. When necessary, or when it presented a better performance, we used native support to EPICS in MXCuBE, implemented using PyEPICS library inside HardwareRepository layer.

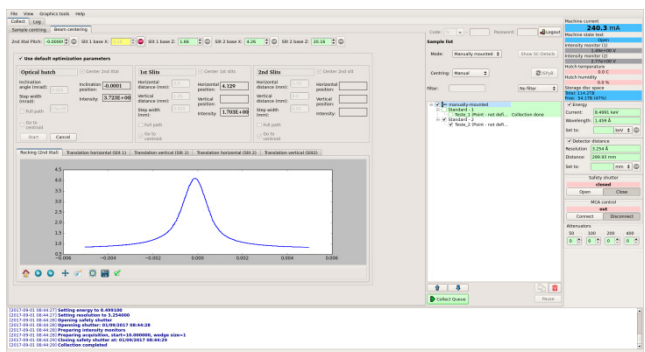

Figure 3: LNLS created widget (brick) to optimize beam position.

Other original features, proposed by technicians of MX2 beamline, were also implemented. One of them is a sequence of beam optimization, to be performed automatically, based on some parameters, like initial and final position of slits, step size, etc. The procedures are

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA201

performed to optimize the energy, scanning the crystal position and calculating energy result based on *Bragg's Law*, and to centralize the beam on two sets of slits, scanning their positions, but using the predefined horizontal and vertical gaps. A photodiode is used to measure beam intensity. Such tool was created as a widget, and the Figure 3 shows the final result.

A CBF viewer embedded in MXCuBE solution was also implemented, using Python *cbf* [6] library developed by *Paul Scherrer Institute (PSI)*. A screenshot of final result is showed in Figure 4.

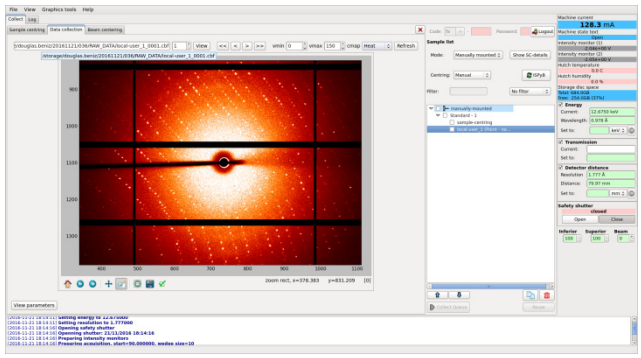

Figure 4: CBF viewer embedded in MXCuBE.

Another example of LNLS implementation inside MXCuBE is the monitoring of dead-time via *Amptec MCA* (Multi-Channel Analyser), what is showed in Figure 5.

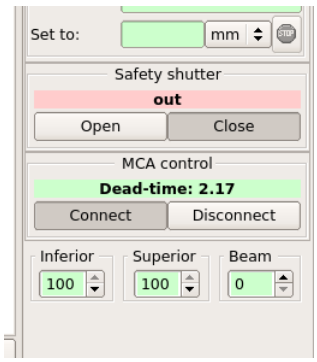

Figure 5: Amptek MCA dead-time

As a last example of original implementation on MXCuBE solution used by MX2 beamline of LNLS, it was implemented an automatic verification of CamServer execution on Pilatus server. Using *Xpra* [7] tool and *wmctrl* Linux command, at MXCuBE start it is verified if CamServer is running on Pilatus server, and if Pilatus EPICS IOC is connected to it, starting it via Xpra, that remotely start GUI based applications, and at the end we stop CamServer using Xpra and wmctrl command, that send commands to X windows. It is displayed at Figure 6 how we can see the CamServer running remotely on Pilatus server via Xpra on operation station where MXCuBE is being executed. Such screen is saved together CBF results on storage.

CONCLUSION AND NEXT STEPS

Since MXCuBE is being used by researchers of MX2 beamline, available to them since 2016's end, we are receiving good feedback from them.

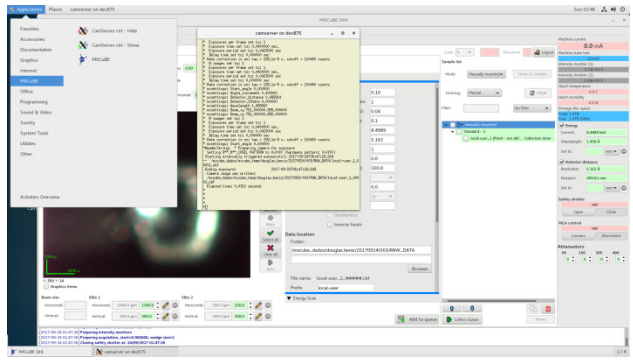

Figure 6: CamServer remotely controlled via Xpra.

Next steps on our macromolecular crystallography beamline, focusing of enhancements for Sirius, are:

- Conclude sample changer programming and installation.
  >Actions already in course, to program Stäubli CS8 robot;
- Enable mesh scan on MXCuBE;
- Enable auto-analysis, at least one first approach, on MXCuBE.
  >Edna is being studied;
- Install ISPyB [8] and integrate it with MXCuBE;

REFERENCES

[1] Gabadinho, J. *et al.*, 2010, "MXCuBE: a synchrotron beamline control Environment Customized for Macromolecular Crystallography Experiments". J. of Synchrotron Rad., V. 17, pp. 700-707.
[2] H. H. Slepicka *et al.*, 2015, "Py4Syn: Python for synchrotrons". J. of Synchrotron Rad., V. 22, pp. 1182-1189.
[3] T. M. McPhillips *et al.*, 2002, "Blu-Ice and the Distributed Control System: software for data acquisition and instrument control at macromolecular crystallography beamlines". J. of Synchrotron Rad., V. 9, pp. 401-406.
[4] J.Hatje, M.Clausen *et.al.*, "Control System Studio (CSS)", ICALEPCS'07, Knoxville, USA, 2007, MOPB03.
[5] AravisGigE driver, `https://github.com/AravisProject/aravis`
[6] cbf package, `https://github.com/paulscherrerinstitute/cbf`
[7] Xpra, `https://www.xpra.org`
[8] ISPyB, `http://www.esrf.eu/ispyb`