USING TKINTER OF PYTHON TO CREATE GRAPHICAL USER INTERFACE (GUI) FOR SCRIPTS IN LNLS

D. B. Beniz†, A. M. Espindola‡, Brazilian Synchrotron Light Laboratory, Campinas, Brazil

Abstract

Python is being widely used to create scripts which cover different necessities in computational scenario. At LNLS (Brazilian Synchrotron Light Laboratory) we successfully developed Python scripts to control beamlines operations, including a case of GUI (*Graphical User Interface*) creation using Tkinter [1] for one of LNLS beamlines, DXAS (*Dispersive X-ray Absorption Spectroscopy*) [2]. In this article its motivation and some implementation details will be presented.

MOTIVATION

The decision to use Python to build a GUI was based on the previous experience with such programming language in LNLS. There is a library package, called Py4Syn [3], developed in LNLS with Python version 3.4 and in use to control beamline devices, like motors and detectors, and to operate a sequence of actions to perform specific experiments by synchronization of a set of such devices and storing of collected data into files formatted in columns to facilitate their analysis. Controllers of such devices have software abstractions, IOCs (*Input/Output Controllers*), developed in EPICS (*Experimental Physics and Industrial Control System*) [4] which resources, PVs (*Process Variables*), are available in the laboratory network via CA (*Channel Access*) [5] protocol. Python offers packages to control such resources, with PyEPICS [6], to perform mathematical calculations and data matrix manipulation, with NumPy [7], and to display data as graphics, with Matplotlib [8], this way it facilitated the development of Py4Syn.

Once we had the tool to elaborate scripts to orchestrate synchrotron beamline experiments, Py4Syn, the new challenge was to offer a GUI that helped users to inform scripts parameters, control and monitor their execution. LNLS adopted CS (Control System) Studio [9] as the tool to monitor and operate EPICS IOCs. It is a great option to monitor and interact with EPICS PVs, however, it is not recommended by their developers to control complex scripts in Python. We tried to use CS Studio to control Py4Syn scripts but the performance was unsatisfactory. Then, we decided to build a GUI in Python, once the scripts were written using that language, and Tkinter arose as a good start as it is the standard GUI package of Python and we found a large number of tutorials and code examples in the Internet. The first experience in LNLS with Tkinter to build graphical interfaces for Python scripts was with DXAS beamline, which was being reformed between 2014 and 2015.

† douglas.beniz@lnls.br
‡ alexey.espindola@lnls.br

ARCHITECTURE OVERVIEW

In Figure 1 the overview of current software architecture solution for control system of DXAS is presented.

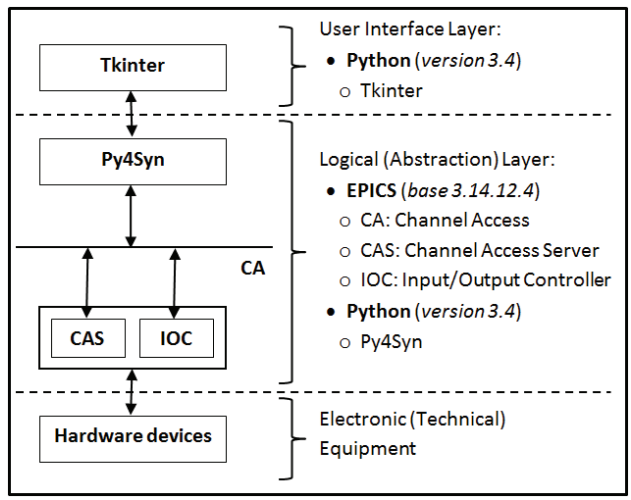

Figure 1: Overview of DXAS Solution Architecture.

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 DXAS beamline of LNLS are:

- Galil DMC-4183: motor controller
- Parker OEM750: motor controller
- Heidenhain MT 2501: optical encoder
- Keithley 6485: picoammeter
- Kepco BOP: power supplier
- OMRON E5CK: digital controller of furnace
- LakeShore 331: temperature controller of cryogenic cooling system
- Stanford SR570: low noise current preamplifier
- Princeton Instruments PyLoN: CCD camera

Logical (Abstraction) Layer

Over the devices controllers is the first abstraction of them, build in EPICS, with correspondent IOCs 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, where each PV is a record, or piece of data, with some attributes to format, configure or simply return related information.

ISBN 978-3-95450-189-2

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 sample, or to vary a furnace temperature while a CCD is acquiring spectra images to analyse sample absorption of x-ray, 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, the top layer of this architecture is the GUI. The focus of this article is the graphical interfaces for scripts that perform automation of procedures to execute experiments in DXAS beamline of LNLS. Such GUIs were implemented, and are being used to operate the beamline since the beginning of 2016, using, mainly, Tkinter package of Python.

The main finalities of such interfaces are to receive parameters and to control the flow of experiment operations. Parameters can be those necessary to configure devices, like the current/voltage of a power supplier of a magnet coil, or the temperature stages (amplitude and duration) of a furnace, or parameters to define initial and final conditions of each experiment, like interval of motor positons to scan, or beam energy amplitude to perform the experiment, or energy interval to scan. Flow operations involve actions of start, pause, when applicable, or stop the experiment.

Some interfaces also show information of some devices that are monitored in short intervals, updated each 100 ms, e.g., like current furnace temperature, power supplier voltage, motor position, among others.

TKINTER CONCEPT

Tkinter, or "Tk interface", is a module of python that provides an interface to Tk GUI toolkit, developed in TCL (*Tool Command Language*) and multiplatform, with support for Linux, MAC OS and MS Windows. Tk is natively present in Linux and MAC OS, and can be easily installed on MS Windows, it is not part of Python. Tkinter is part of Python, being called "Tkinter" in versions prior to 3, and "tkinter" on version.

Widgets, geometry management and event handling are the three main concepts of Tk, which also apply for Tkinter.

Widgets

Often referred to as controls, or window elements, widgets are all visible components on a graphical interface. Some examples are frames, labels, buttons, text entries, checkboxes, tree views, scrollbars, and text areas.

Inside a Python script, widgets are objects, or instances of classes that represent mentioned window components. To instantiate an object on Tk, and then on Tkinter, is necessary to indicate its parent, what maintain a window hierarchy between all elements. On that hierarchy, the main window is the root. Each widget has a set of configuration options which control how they are displayed or how they behave, like a "text" option for components that display some text, as a label, or "command" option when they accept events, as the mouse click of a button.

Geometry Management

An important step of interface design is to organize the widgets onscreen window. The most useful method to do that using Tk, or Tkinter, is by a geometry manager, like "grid". In practice, "grid()" is a method available to all supported widgets saying to then where exactly to be positioned in an invisible matrix of columns and rows.

Combination of nested frames and grid is the better approach to design a Tk/Tkinter interface.

Event Handling

Tk/Tkinter manages the event loop that receives user actions over the window components, controlled by operating system, like button presses, keystrokes, mouse movement, and window resizing.

Individual widgets know how to respond to events. Basically, it provides a callback that can be assigned to a procedure in Python code as a configuration, like "command" for button widgets. For events without a callback command associated with them, it is possible to use an event binding, which in practice is the use of "bind()" method on a widget to capture any event and then execute an arbitrary procedure or method.

Another important method available for widgets is "after()". Using it is possible to create an execution thread forked from the main application loop. At this new thread, a set of Python instructions is performed in parallel with interface updating. Besides, widget that calls it continues to be responsive to any user input.

TKINTER SOLUTION OF DXAS

For DXAS beamline of LNLS the main techniques supported are detection of very weak signals XANES (*X-ray Absorption Near-Edge Spectroscopy*), XAFS (*X-ray Absorption Fine Structure*), XMCD (*X-ray Magnetic Circular Dichroism*), XRMS (*X-ray Resonant Magnetic Scattering*), Catalysis and Cryogenics experiment (analysis of *X-ray Absorption* during a very high or very low temperature exposition).

So, the graphical interfaces built for DXAS control system solution cover these operations:

- **Spectroscopy**
- **X-ray Absorption**
- **XMCD / XRMS**
- **Catalysis / Cryogenics**
- **Motor Scan**

ISBN 978-3-95450-189-2

And some specific operations that include devices monitoring:

- **Slits** (with virtual motors built by Py4Syn conjugating real motors)
- **E5CK** (with temperature ramp programming and furnace monitoring)
- **Kepco BOP** (to set and monitor power supplier configuration and amplitude)
- **Generic graphics plot** (to plot graphics from any supported file by this solution)
- **Generic data consolidation** (to calculate results for all supported files generated by this solution)

Figure 2 below shows an excerpt of source code of XMCD and XRMS interface. On that, it is possible to see a LabelFrame and some Button widgets being instantiated, with their configurations being set, as described above in "Tkinter Concept" session.

magnetic_dxas.py

```
236
237 # ------------------------------------------------------------
238 self.buttonsFrame = LabelFrame(self.master, text = 'Run experime
239 self.buttonsFrame.pack(side = TOP, fill = X, pady = 2, ipadx = 2
240 self.startButton = Button(self.buttonsFrame,
241                           command = self.StartExperiment,
242                           text = "Scan",
243                           font = textFont4,
244                           background = "green",
245                           fg="black",
246                           state = DISABLED,
247                           height = 1,
248                           width = 8)
249 self.startButton.pack(side = LEFT, anchor = W, padx = 10, pady =
250 self.stopButton = Button(self.buttonsFrame,
251                           command = self.StopExperiment,
252                           text = "Cancel",
253                           font = textFont4,
254                           background = "red",
255                           fg="white",
256                           state = DISABLED,
257                           height = 1,
258                           width = 8)
259 self.stopButton.pack(side = LEFT, anchor = W, padx = 10, pady =
```

Figure 2: Code Excerpt of XMCD and XRMS interface.

Figure 3 shows another code excerpt, this time of E5CK interface. Here we see the mechanism of instantiating an independent thread to update furnace temperature on the UI after each 1 second, as described above in "Tkinter Concept" session.

*e5ck.py

```
231
232 def callback_curr_temp():
233     try:
234         self._currTemp.set(round(self.furnaceDevice.getValue(),
235         self._currPower.set(round(self.furnaceDevice.getPower()
236         self._currP.set(round(self.furnaceDevice.getP(),2))
237         self._currI.set(round(self.furnaceDevice.getI(),2))
238         self._currD.set(round(self.furnaceDevice.getD(),2))
239     except (ValueError, TypeError):
240         pass
241     global processRunning
242
243     # Updates total time
244     try:
245         totalFurnace = self.__get_total_program_time()
246         if (self._radioTime.get()==0):
247             totalFurnace = totalFurnace * 60
248         self.Furnace_Table._totalTime.set(str(round(totalFurnac
249     except:
250         pass
251
252     # And update it every 1 second later
253     self.master.after(1000, callback_curr_temp)
254 # Trigger of callback (point of return)
255 self.master.after(1000, callback_curr_temp)
256
```

Figure 3: Code Excerpt of E5CK interface.

Finally, Figure 4 gives an example of how GUI for DXAS beamline control system solution implemented using Python Tkinter looks like.

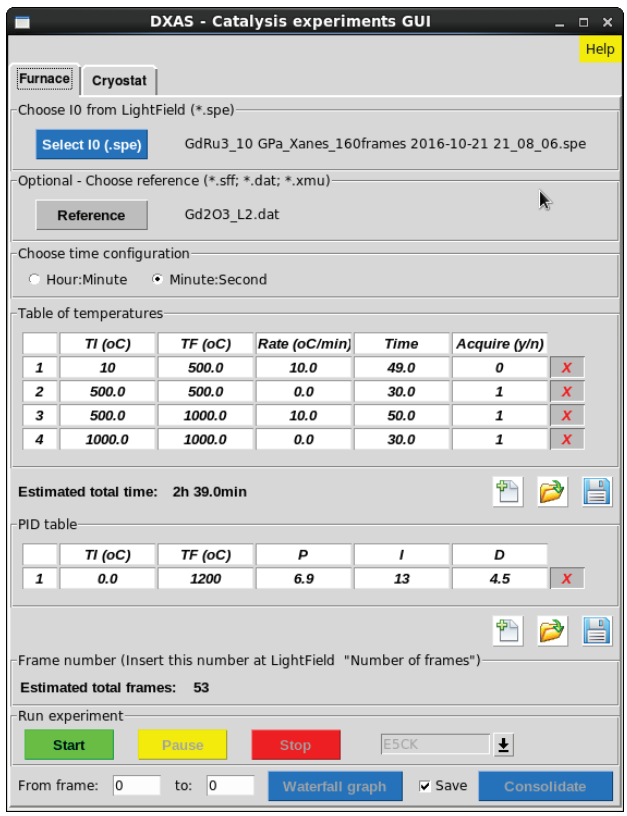

Figure 4: GUI of Catalysis Experiments.

REFERENCES

[1] Tkinter, https://wiki.python.org/moin/TkInter

[2] DXAS beamline of LNLS, http://lnls.cnpem.br/beamlines/xafs/beamlines/dxas/

[3] Py4Syn, http://py4syn.readthedocs.io/en/latest/

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

[5] CA, http://www.aps.anl.gov/epics/docs/ca.php

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

[7] NumPy, http://www.numpy.org/

[8] Matplotlib, http://matplotlib.org/

[9] CS-Studio, http://controlsystemstudio.org/

ISBN 978-3-95450-189-2