EFFORTLESS CREATION OF CONTROL & DATA ACQUISITION GRAPHICAL USER INTERFACES WITH TAURUS

C. Pascual-Izarra#, G. Cuní, C. Falcón-Torres, D. Fernández-Carreiras, Z. Reszela, M. Rosanes,
ALBA-CELLS Synchrotron, Cerdanyola del Vallès, Spain
T. Coutinho, ESRF, Grenoble, France

Abstract

Creating and supporting Graphical User Interfaces (GUIs) for experiment control and data acquisition has traditionally been a major drain of time and resources for laboratories. GUIs often need to be adapted to new equipment or methods, but typical users lack the technical skills to perform the required modifications, let alone to create new GUIs. Here we present the Taurus [1] framework which allows a non-programmer to create a fully-featured GUI (with forms, plots, synoptics, etc.) from scratch in a few minutes using a "wizard" as well as to customize and expand it by drag-and-dropping elements around at execution time. Moreover, Taurus also gives full control to more advanced users to access, create and customize a GUI programmatically using Python [2]. Taurus is a free, open source, multi-platform pure Python module (it uses PyQt [3] for the GUI). Its support and development are driven by an active and welcoming community participated by several major laboratories and companies which use it for their developments. While Taurus was originally designed within the Sardana [4] suite for the Tango [5] control system, now it can also support other control systems (even simultaneously) via plug-ins.

INTRODUCTION

Taurus is a framework for creating user interfaces (both GUIs and command-line based) to interact with scientific and industrial control systems as well as with other related data sources.

In this work we first give a brief overview of Taurus and then we focus on one of its key assets: the possibility of deploying powerful, customizable and flexible control and data acquisition GUIs within minutes without requiring to write a single line of code. Finally, we discuss the current development efforts and the future plans.

OVERVIEW

Background

Taurus was originally conceived (under early internal names such as Tauico, Tauiwi, and Tau) as the ALBA [6] synchrotron's in-house solution for connecting client side applications to Tango device servers [7]. It provided the user interface code for the Sardana suite [8], which is used in ALBA for control and data acquisition of both the accelerator and all the beamlines.

After its first public release in 2011, Taurus has been adopted by several large laboratories and companies, and it has become very popular among many newcomers to the Tango Collaboration[1].

Since 2012 much effort has been put into bringing control system agnosticity into Taurus: proof-of-concept plugins for supporting EPICS [9] and SPEC [10] were developed and the Taurus core has been refactored in order to isolate the Tango dependencies into an optional plugin for the next major release (Taurus 4)[2].

Community

The Taurus Community largely intersects with the Sardana Community (from which it spun off to reflect the fact that Taurus can be used independently of Sardana) and shares its organizational characteristics and open development model [11, 12]: public code review process, proposal-driven decision taking [13], periodical meetings (~yearly), free licensing (LGPLv3+ [14]), etc.

FAST GUI CREATION

Model-View-Controller Approach

Taurus uses the Model-View-Controller (MVC) pattern [15] to build interfaces[3].

The *taurus.core* module uses plugins (known as *schemes*) to provide *TaurusModel* objects that abstract the interactions with specific sources of data and/or control objects. Some schemes are already implemented for accessing control system libraries (the “tango”, “epics” and “spec” schemes) as well as for data-processing via a Python interpreter (the “*evaluation*” scheme).

Every TaurusModel object can be of type *Authority*, *Device* or *Attribute*, and has a unique name in the form of a Unified Resource Identifier (URI). See Table 1 for some examples of model names. Each scheme implements a

#cpascual@cells.es

[1] While early adopters of Taurus were mostly synchrotrons (*Desy, MAX-IV, Solaris, ESRF,...)* other scientific institutions such as the *ELI-ALPS* and *LULI-APOLLON* large laser installations or the *ONERA* wind tunnel are currently using it. Companies such as *Cosylab*, *Nexeya*, *Tata Consultancy Services* and *Observatory Sciences are* providing services based on Taurus to a growing number of institutions, including the world largest radio telescope (SKA).

[2] Unless explicitly stated otherwise, in the rest of the article the situation described corresponds to the current Taurus4 state.

[3] Many Taurus components combine the View and Controller roles. Those cases could be referred to as “Model-View” instead of MVC.

ISBN 978-3-95450-148-9

Table 1: Taurus Model Name Examples

#	Model name (URI)	Model type	Scheme	Represented source of data/control object
1	*tango://foo:1234*	Authority	tango	Tango database listening to port *1234* of host *foo*
2	*tango://foo:1234/a/b/c*	Device	tango	Tango Device *a/b/c* registered on the above database (#1)
3	*tango:a/b/c/state* [(1)]	Attribute	tango	*state* attribute of Device #2 when database #1 is the default [(1)]
4	*tango://foo:1234/a/b/c/d* (or *tango:a/b/c/d* [(1)])	Attribute	tango	Tango Attribute *d* of device #2 (and its implicit DB form [(1)])
5	*epics:XXX:m1.VAL* [(1)]	Attribute	epics [(2)]	EPICS process variable *XXX:m1.VAL*
6	*eval:({tango:a/b/c/d}+{epics:XXX:m1.VAL})*0.5* [(1)]	Attribute	evaluation	Calculated average of the values of #4 and #5
7	*eval:rand(256)* [(1)]	Attribute	evaluation	Random generated array of 256 values
8	*msenv://foo:1234/macroserver/bar/1/ScanDir*	Attribute	msenv [(3)]	ScanDir environment variable from Sardana's msenv scheme
9	*h5file:/mydir/myfile.hdf5* [(1)]	Device	h5file[(3)]	File in HDF5 format saved at */mydir/myfile*
10	*h5file:/mydir/myfile.hdf5:data/energy* [(1)]	Attribute	h5file[(3)]	HDF5 dataset *energy* of group *data* from file #9
11	*ssheet:myfile.ods:Sheet1.A1* [(1)]	Attribute	ssheet[(3)]	Contents of cell A1 of Sheet1 of *myfile.ods* spreadsheet

Notes:
[(1)] Some parts of the model URI (e.g. the authority segment) may be omitted if default values are defined.
[(2)] The epics scheme implementation is still a proof-of-concept. URI syntax may vary in its final implementation.
[(3)] These schemes are only in discussion stage. URI syntax may vary when implemented.

Factory object that takes model names and returns the corresponding TaurusModel objects.

The Taurus view and controller components can be implemented in many forms: command line interfaces such as Sardana's *spock*, web based applications such as those demonstrated in the *taurus.web* module (just a proof of concept for now) or, the most common, PyQt based GUIs.

The *taurus.qt* module provides a set of basic widgets (labels, LEDs, editors, forms, plots, tables, buttons, synoptics,...) that extend related Qt widgets with the capability of attaching to Taurus core models in order to display and/or change their data in pre-defined ways. For example, a *TaurusPlot* widget will display a curve for each attribute model to which it is attached if its value is a one-dimensional numerical array. Similarly, a *TaurusForm* widget will allow the user to interact with the data represented by its attached models (Fig. 1). The actual association of a view (widget) with a model is done by providing the model name to the widget.

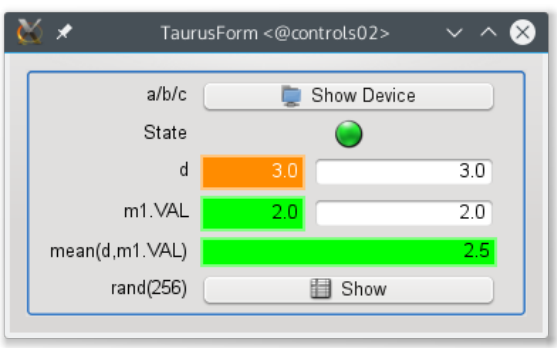

Figure 1: TaurusForm widget attached to models from tango, epics and python evaluation schemes (#2, #3, #4, #5, #6 and #7 from Table 1).

Note that thanks to the model abstraction provided by the scheme plugins, Taurus based applications can transparently mix data from different sources, as demonstrated in Fig. 1.

TaurusGui vs *Qt Designer*

The Taurus widgets behave just as any other Qt widget, and as such, developers could use them to create GUIs in a regular way, both programmatically or using the Qt designer (Taurus extends the Qt designer catalogue with its own widgets).

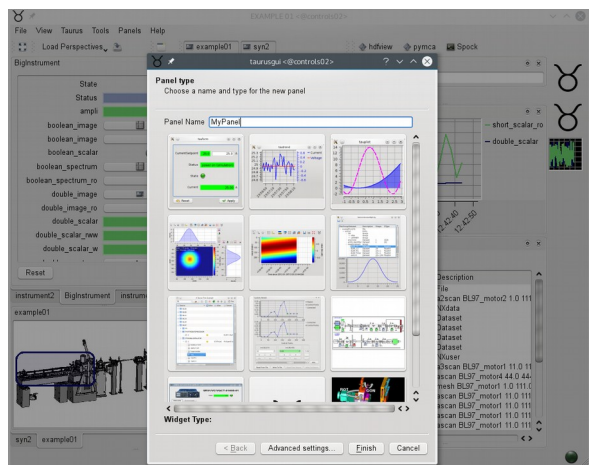

Figure 2: TaurusGUI (in the background) showing the "*New Panel Catalogue*" in the foreground.

However, Taurus provides an even simpler and much more dynamic way of creating GUIs: the *TaurusGUI* framework, which allows the users to create a skeleton of a GUI with a few clicks on a wizard and then populate it at run time with *panels* containing any arbitrary widget from the Taurus catalogue or from an external module (see Fig. 2). The layout of the panels in the application is

ISBN 978-3-95450-148-9

completely customizable (they can be dragged around, stacked, hidden, made into tabs,...), and different view configurations, called *perspectives*, can be saved and retrieved at any moment to adapt to the task or to the user preferences.

The Taurus widgets can be associated with models when they are added to the GUI and, in many cases, they may also accept drag & drop of the model name(s) from other widgets (or from a model selector) at any time, e.g.: the user can start plotting the time evolution of a given value by just dragging its name from a TaurusForm and "dropping" it into a *TaurusTrend* widget.

TaurusGUIs can be modified (e.g., an extra plot widget may be added on-demand) without coding and users even create whole new temporary TaurusGUIs from scratch to solve particular tasks.

Simplicity for Users, Control for Experts

All the convenience described above does not limit the control that experts may want for tweaking and adapting the system to their needs. This extra control can be exerted at several levels:

First, it is possible to edit the *configuration files* that define a TaurusGUI-based application. These are declarative python and XML files (editable as plain text) complemented by Qt settings files (editable with the provided *taurusconfigbrowser* application).

On a lower level, custom specific widgets (created either programmatically or via the Qt designer) can be added as panels to a TaurusGUI application. At this level, it is also possible to do simple inter-panel communication thanks to the *SharedDataManager* broker component provided by the TaurusGUI framework (see *broker pattern* in [15]). This is used by many Taurus-based GUIs such as VACCA [16] to provide synoptic-based GUI navigation: the visual representation of an instrument in a synoptic can be clicked to show the panel associated with it and, conversely, the synoptic element gets highlighted if its associated panel is selected.

Finally, the maximum level of control can be achieved by programmatically accessing the TaurusGUI class itself. In this way, all the higher level features described before are still available, while there are no limitations on the customizations that can be done.

CURRENT & FUTURE DEVELOPMENTS

Taurus 4.x

At the moment of writing, the latest production-ready release of Taurus is at the 3.6 version, and the next major version (Taurus 4.x) which is currently under development (the core is ready and the widgets are being adapted to it), is planned to be released soon.

Taurus4 implements the TEP3 and TEP14 enhancement proposals [13], simplifying the Application Programming Interface (API) of the core, e.g.: the distinction between an Attribute and its Configuration disappears, many redundant methods are deprecated and the model name syntax is made RFC3986-compliant [17]. This facilitates the creation of new scheme plugins. Also, as part of this refactoring, all the dependencies on PyTango, as well as many tango-influenced APIs have been isolated in the tango scheme plugin, making it possible to run Taurus on a machine without Tango.

Another key improvement from TEP14 is the support for measurement units: all numerical values from the models representing a physical quantity (e.g., the read value, the limits, etc.) will be *Quantity* objects from the *Pint* module [18], enabling user-friendly unit conversions and enforcing dimensional consistency when operating with them.

Taurus4 also brings a significant leap in unit test coverage since many of its features have been implemented using test-driven development.

One of the main concerns when implementing the above mentioned changes was to maintain the backwards-compatibility with the previous version (i.e., that programs developed for Taurus 3.x should ideally work with 4.x). This has been achieved to a large degree by implementing a backwards-compatibility API that translates 3.x API calls to their 4.x equivalents while issuing deprecation warnings). Only in a few cases it has not been possible to implement such an automatic transition (mostly related to tango-centric APIs that cannot be generalized to other schemes), and in these cases a deprecation error is raised. We expect that most GUIs will be able to transition to 4.x without modification (with possibly some non-critical warnings that can be solved at a later moment). The rest may need some minor modifications to run on 4.x, but as long as the basic Taurus widgets are not heavily modified, the update effort should be moderate.

Technology Stack Update

Keeping Taurus up-to-date with the evolution of the technologies is a continuous effort that allows us to guarantee the maintainability of Taurus.

At this moment Taurus is compatible with Python 2.6 and 2.7. We intend to add support to Python 3 too (probably we will drop the support for 2.6 to facilitate maintaining a common codebase for 2.7 and 3.x).

Similarly, we currently support PyQt 4.4 and newer but not PyQt 5.x or PySide since we are limited by PyQt 4.4 to use old-style signals. In the near future, we intend to adopt new-style signals and support all PyQt versions from 4.8 (5.x included) as well as PySide.

Regarding graphics visualization, the *plot* and *extra_guiqwt* Taurus modules[4] depend on the no longer

[4] *taurus.qt.qtgui.plot* provides 2D plots and trends, while *taurus.qt.qtgui.extra_guiqwt* provides contour/colour plots and image visualization as well as alternative implementations of the 2D plots and trends

ISBN 978-3-95450-148-9

maintained PyQwt 5.2 module [19]. The Taurus Community is currently discussing about the best alternatives for the future of the graphics in Taurus. The options being considered are: a) base all the plotting on *guiqwt* [20], or b) to contribute to the incipient PythonQwt module [21] to bring it to a point where it is a viable drop-in replacement for PyQwt in Taurus, or c) start a new plotting infrastructure based on PyQtGraph [22] or PyMca [23] (which would require more initial effort but would open the possibility to provide OpenGL-based 3D data visualization).

Unification of Extension APIs

Another pressing priority in Taurus is the implementation of a formal API for providing extensions (plugins). At this moment, different mechanisms to support extensions are already implemented in various subsystems of Taurus, such as the schemes in the core, the widgets in *taurus.qt.qtgui*, the panel catalogue in TaurusGUI, the icons, the extension API in the tango factory, etc. (see the TEP13 [13] for more details). Most of these mechanisms, being ad hoc implementations, are quite specific and present limitations such as requiring the plugin to have privileged access to Taurus installation directories, or not having a well defined interface, or not managing dependencies/incompatibilities among plugins.

This situation will be improved by adopting a generic extension mechanism (e.g., stevedore [24] or yapsy [25]) and using it throughout the whole Taurus library. Apart from facilitating the improvement and maintainability of the code (removing multiple different implementations and APIs), the Taurus library will become simpler to extend and lighter, since many sub-packages that are currently monolithic may be reimplemented as a collection of optional extensions to be installed and/or loaded on-demand.

As a side-effect, other projects currently extending Taurus like Sardana, VACCA or PANIC [26] will also benefit from a supported extension API in Taurus: first, by formally registering themselves as Taurus extensions and second, by internally using the same API for their own plugins (e.g., the macros, controllers and recorders in Sardana).

From a community point of view, we expect that the increased modularization resulting from this change will facilitate collaborations since external developers will find it easier to experiment and contribute their changes with less worries of breaking critical components.

Multi-Model API

Another foreseen improvement in Taurus is to extend the existing API for attaching widgets to models. The current API only supports one model per widget, and those widgets requiring to attach to more than one model need to improvise ad hoc solutions. A proposal based on using class decorators to provide multi-model support is being evaluated.

New Schemes

The core refactoring done in Taurus 4 has been the main blocker for the creation of new schemes. Once Taurus 4 is released, we expect that several schemes will be (re)implemented: apart from those mentioned as incomplete in Table 1 (*epics*, *h5file*, *msenv*, *ssheet*), other schemes have been proposed to support the following data sources: the MADOCA-II control system [27]; LIMA devices [28]; archiving systems [29]; SQL databases; plain text files containing tabular data, etc.

CONCLUSION

Our experience shows that the TaurusGUI framework greatly improves the way that control and data acquisition interfaces are deployed: users (even the non-programmer ones) create and/or customize their own GUIs, gaining in autonomy and minimizing both their waiting time and the burden on support engineers.

Taurus is already a reference within the Tango community, but its full potential outside it (both in large and small installations) is still open to be explored and will be enabled by the release of Taurus 4.

ACKNOWLEDGEMENT

We would like to thank the Taurus, Sardana and Tango community members and the ALBA Controls Group and, specially, to Teresa Nuñez (Desy), Jan Kotanski (Desy), Valentin Valls (ESRF) and Sergi Rubio (ALBA) for their contributions to Taurus. We would also like to thank Frederic Picca (Soleil) for his suggestions and for packaging Taurus and Sardana for Debian. Finally, we are indebted to all those who participated in the seminal discussions that gave birth to Tau.

REFERENCES

[1] Taurus website: http://www.taurus-scada.org
[2] Python website: http://www.python.org
[3] PyQt website: http://www.riverbankcomputing.com/software/pyqt/
[4] Sardana website: http://www.sardana-controls.org
[5] Tango website: http://www.tango-controls.org
[6] ALBA website: http://www.albasynchrotron.es
[7] D. Fernandez-Carreiras et al., "ALBA, a Tango Based Control System in Python", THP016, ICALEPCS2009, Kobe, Japan, (2009).
[8] T. Coutinho et al. "Sardana, the Software for Building SCADAs in Scientific Environments", WEAAUST01, ICALEPCS2011, Grenoble, France, (2011).
[9] EPICS website: http://www.aps.anl.gov/epics/
[10] SPEC website: http://www.certif.com/content/spec/
[11] Z. Reszela et al., "Sardana – a Python Based Software Package for Building Scientific SCADA Applications", WCO206, PCaPAC2014, Karlsruhe, Germany, (2014).

ISBN 978-3-95450-148-9

[12] Z. Reszela et al., “Bringing Quality in the Controls Software Delivery Process”, MOPGF171, ICALEPCS2015, Melbourne, Australia, (2015).
[13] Taurus Enhancement Proposals: http://sf.net/p/tauruslib/wiki/TEP/
[14] Lesser General Public License: https://www.gnu.org/licenses/lgpl.html
[15] F. Buschmann et al., Pattern-oriented software architecture: a system of patterns, (New York: Wiley, 1996), 125.
[16] S. Rubio-Manrique et al., “Unifying All TANGO Control Services in a Customizable Graphical User Interface”, WEPGF148, ICALEPCS2015, Melbourne, Australia, (2015).
[17] Berners-Lee et al., "Uniform Resource Identifiers (URI): Generic Syntax". Internet Engineering Task Force. http://tools.ietf.org/html/rfc3986
[18] Pint website: http://pint.readthedocs.org
[19] PyQwt website: http://pyqwt.sourceforge.net
[20] guiqwt website: https://pythonhosted.org/guiqwt/
[21] PythonQwt website: http://pythonhosted.org/PythonQwt/
[22] PyQtGraph website: http://pyqtgraph.org
[23] V.A. Solé et al., A multiplatform code for the analysis of energy-dispersive X-ray fluorescence spectra, Spectrochim. Acta Part B 62 (2007) 63-68.
[24] stevedore website: http://docs.openstack.org/developer/stevedore/
[25] yapsy website: http://yapsy.sourceforge.net/
[26] S. Rubio-Manrique et al., “PANIC, a suite for visualization, logging and notification of incidents”, FCO206, PCaPAC2014, Karlsruhe, Germany, (2014).
[27] T. Matsumoto et al., “Next-Generation MADOCA for the SPRING-8 Control Framework”, TUCOCB01, ICALEPCS2013, San Francisco, USA, (2013).
[28] A. Homs et al., “LIMA: a Generic Library for High Throughput Image Acquisition”, WEMAU011. ICALEPCS2011, Grenoble, France, (2011).
[29] S. Rubio-Manrique et al, “Validation of a MySQL based archiving system for Alba Synchrotron”, WEP010, ICALEPCS2009, Kobe, Japan, (2009)

ISBN 978-3-95450-148-9