ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

TAURUS BIG & SMALL: FROM PARTICLE ACCELERATORS TO DESKTOP LABS

C. Pascual-Izarra†, G. Cuní, C. Falcón-Torres, D. Fernández-Carreiras, Z. Reszela, M. Rosanes,
ALBA-CELLS Synchrotron, Cerdanyola del Vallès, Spain
Oscar Prades-Palacios, ETSE-UAB, Cerdanyola del Vallès, Spain

Abstract

Taurus is a popular solution for rapid creation of Graphical User Interfaces (GUIs) for experiment control and data acquisition (even by non-programmers) [1]. Taurus is best known for its ability to interact with the Tango and Epics control systems, and thus it is mainly used in large facilities. However, Taurus also provides mechanisms to interact with other sources of data, and it is well suited for creating GUIs for even the smallest labs where the overhead of a distributed control system is not desired. This scalability together with its ease-of-use and the uncontested popularity of Python among the scientific users, make Taurus an attractive framework for a wide range of applications. In this work we discuss some practical examples of usage of Taurus ranging from a very small experimental setup controlled by a single Raspberry Pi, to large facilities synchronising an heterogeneous set of hundreds of machines running a variety of operating systems.

INTRODUCTION

Taurus [2] 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. Its main strength is that fully-functional GUIs can be created with minimum effort even by non-programmers, while still allowing full control and the possibility of extending its capabilities by more advanced developers.

Taurus is a free, open source, multi-platform pure Python module (it uses PyQt [3] for the GUI). It uses a Model-View approach to building the GUIs where the complexities of lower-level access to the data sources or control libraries is abstracted away by a set of plugins that provide Taurus model objects. The graphical components of a GUI just need to be provided with one or more model names in order to display and/or control the data represented by the model(s), allowing the creation of fully functional GUIs in a few minutes without programming [1].

These characteristics, combined with the popularity of Python in scientific environments made Taurus the preferred framework for GUI creation at many facilities.

Taurus was originally developed at the ALBA synchrotron within the Tango Collaboration [4] and therefore the first data source to be supported was the Tango Distributed Control System (DCS), but it is not limited to it. Other DCSs such as Epics as well as generic data sources such as "python evaluation" or hdf5 files are supported via "scheme" plugins. Since Taurus version 4, the architecture for new user-contributed scheme plugins was simplified and the Taurus core was made completely scheme-agnostic (i.e. Tango support is implemented just as another scheme plugin). The main widgets provided with Taurus (forms, plots, labels, edit boxes, etc.) are also scheme-agnostic, allowing the implementation of GUIs that integrate one or more sources of data, as shown in Fig. 1.

In the following sections we describe the different strategies for integrating a given data source into Taurus and we discuss how this allows Taurus to be used in a wide range of contexts, from large facilities with thousands of controllable parameters to single-instrument laboratories.

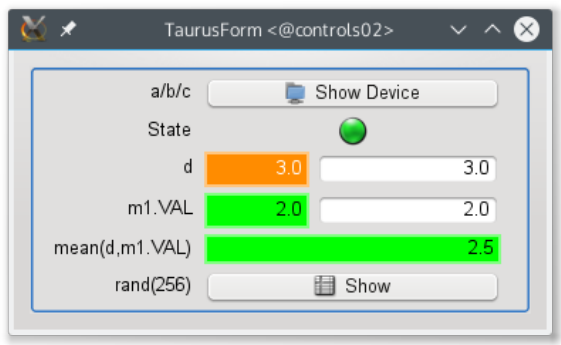

Figure 1: TaurusForm widget attached to models from *tango*, *epics* and *evaluation* schemes.

ACCESSING DATA SOURCES FROM A TAURUS APPLICATION

As already mentioned, the scheme plugins provide Taurus model objects which are used by the Taurus widgets to enable the interaction with the experimental setup and data sources. A scheme plugin implements specific Taurus model objects, which can be of one of three types: Attribute (a model that provides a value and related metadata), Device (a stateful model that may execute actions, or may be a natural aggregator of Attributes) and Authority (a model that provides a context for Devices and Attributes). A Taurus model has a unique name in the form of a Unified Resource Identifier (URI)

† cpascual@cells.es

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

Table 1: Examples of Usage of the *Evaluation* Scheme

#	Model name (URI)	Description
1	*eval:({tango:a/b/c/d}+{epics:XXX:m1.VAL})*0.5*	declares an attribute whose value is the average of the values of a tango attribute and an epics process variable
2	*eval:rand(256)*	declares an attribute whose value is an array of 256 pseudo-random values
3	*eval:Q(rand(256),'V')*	Same as #2, but assigning units (Volts)
4	*eval:@datetime.*/date.today().isoformat()*	uses the *datetime* module to get today's date as a Taurus string attribute
5	*eval:@os.*/environ["TANGO_HOST"]*	accesses a system environmental variable as a string attribute (uses the *os* module)
6	*eval:@os.path.*/getsize("/var/log/boot")<50*	declares a boolean attribute indicating if the size of a file is below a threshold (uses *os* module)
7	*eval:@pandas.*/read_csv('foo.csv')['y'].as_matrix()*	reads a column of a CSV file as an array attribute (uses *pandas* module)
8	*eval:@c=cv2.VideoCapture(0)/c.read()[1][...,1]*	declares an array attribute from captured images from a webcam (uses *opencv* module)
9	*eval:@c=mymod.MyClass()/c.foo*	declares an attribute from a member (*foo*) of an instance of a custom class (*mymod.MyClass*). Note that if *foo* is a writable python property, the resulting attribute would also be writable.

and the scheme plugin provides name validators for each type of model. The scheme also provides a Factory object that takes model names and returns the corresponding model objects.

When facing the integration of an instrument or a data source into a Taurus application one should check if an existing scheme already supports it. If that is not the case, there are three possibilities:

a) integrate the new instrument into a supported DCS and then access it via the corresponding DCS scheme.

b) writing a specific scheme

c) using the generic *evaluation* scheme

Integrating the new instrument into a supported DCS (e.g., writing a new Tango Device Server or an Epics Driver) makes sense when the person in charge is experienced with the given DCS. This is usually the preferred choice when the new instrument will be part of a larger experimental setup already controlled by the DCS.

On the other hand, writing a custom scheme plugin is a good solution either for isolated setups where a DCS is not desired or for performance reasons when direct access to the data source is more convenient. Writing a custom scheme is also an option for large facilities with an existing DCS currently unsupported by Taurus that are interested in using Taurus to build user interfaces to their own DCS. Implementing a custom scheme involves creating a python module to provide subclasses of the base attribute, device and authority model classes, the name validator classes and the factory class. The effort required to implement it depends enormously on the details of how the data source needs to be accessed (e.g. whether resources need to be protected from concurrent access, optimizations, etc.) as well as on the model features to be supported (e.g. whether the scheme will allow to write data). As an example, a first working implementation of the h5file scheme plugin (read-only access to data in hdf5 files) [5] took about 8h to a developer who was familiar with the Taurus concepts.

Finally, using the generic *evaluation* scheme is usually a quicker option, suitable for prototyping or for accessing a relatively simple experimental setup. The *evaluation* scheme allows to embed arbitrary python code into the model name URI. It is typically used to dynamically create attributes that result from performing mathematical operations on other attributes, but it can also be used to declare Taurus attribute models whose value is the result of evaluating code from arbitrary modules (see Table 1 and Fig. 2). It is therefore possible to quickly integrate some hardware for which a python access module exists (see, e.g. example #8 of Table 1, where a webcam is controlled) without having to write any specific code.

Furthermore, thanks to the support for writeable *evaluation* attributes, it is possible to act on the hardware (not just read from it), although this would typically involve writing a small wrapper code in order to expose the actions as writeable properties (see example #9 of Table 1).

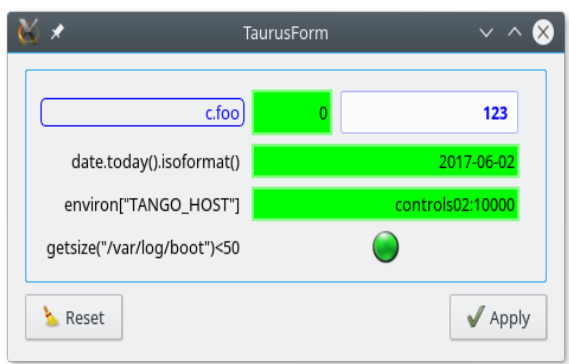

Figure 2: TaurusForm widget demostrating access to arbitray modules (examples 9, 4, 5 and 6 from Table 1).

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

TAURUS FOR LARGE FACILITIES

Due to the already explained historical reasons, the first facilities adopting Taurus were synchrotrons from the Tango community (ALBA, DESY, MAX IV, Solaris, ESRF,...). Nowadays, Taurus is used to some degree also by LASER facilities (ELI, LULI,...), by linear accelerators (LAL, CMAM,...), by a wind tunnel (Onera), by large astronomy infrastructures (SKA), by nuclear research facilities (CEA, Kurchatov Institute,...) among other large laboratories. Also, several companies provide services based on Taurus and/or implement custom solutions with it.

At the moment of writing, we are not aware of Taurus being used by any large facility whose main DCS is different than Tango, but some have expressed interest and evaluated the possibility of supporting their own DCSs (ESO, SPRING-8). Also, Taurus is already used at DESY to integrate Epics-based subsystems within a general Tango system.

In the case of ALBA, Taurus and Taurus-based frameworks such as Sardana [6, 7], Vacca [8], PyAlarm [9] are used across all areas, from controlling the accelerators to acquiring experimental data at all the beamlines. The roughly 100 control GUIs developed in-house are almost exclusively based on Taurus, and they interact with a Tango DCS consisting of tenths to hundreds of thousands attributes distributed amongst more than 300 machines. Both Taurus and Tango are run at ALBA on a heterogeneous set of machine architectures and Operating Systems [10].

The *evaluation* scheme is also heavily used at ALBA, mostly by end-users wishing to customize their own views over the system. The Epics scheme plugin has also been used to integrate a small subsystem, but not in production yet.

From all the accumulated experience, we can conclude that Taurus has demonstrated to work well for large facilities where it scales just as well as the underlying DCS.

TAURUS FOR SMALL LABORATORIES

Since the scalability towards complex systems has already been established, we will now discuss the performance of Taurus in small laboratories (down to "desktop labs" consisting of a single computer connected to a single instrument).

There is one first approach which consists in simply scaling down the solution used in large facilities, i.e., using Taurus on top of a DCS. This is possible because both currently supported DCSs (Tango and Epics) can be easily run even on a single computer. This is certainly a good solution in those situations where: a) the person-in-charge is already familiar with the DCS and b) the instrument hardware is already supported by the DCS (e.g., a Tango Device Server already exists for your equipment). The advantages of this approach are that it simplifies future expansions and that it benefits from the robustness and community support from the DCS.

As an example of this approach, we can mention that installing and configuring Tango+Taurus (including setting-up a Tango Database) on a single PC can be done in a few minutes (e.g. with just one "apt-get" command in Debian). Using a DCS certainly introduces some resources overhead, but this is generally acceptable: e.g., a single RaspBerryPI 3 mini-computer [11] has been shown to be able to simultaneously run a Tango Database, the Sardana Device Server, and a Taurus GUI accessing a microscope.

However, this approach may not be the most convenient if the above-mentioned conditions are not met. Especially if the person responsible for the setup is not already familiar with the DCS: learning all the involved concepts required for configuring and maintaining the software infrastructure required by a DCS may become too much overhead for a small setup whose responsible may not even be familiar with network protocols, system services, etc.

In this case, Taurus benefits from the possibility of being used *without* an underlying DCS. Taurus Model Objects may be provided also by non-DCS scheme plugins such as the *evaluation* scheme, or the *h5file* scheme, or any other custom-created scheme plugin.

As an example of a minimal setup based on non-DCS schemes, consider the case of the system depicted in Fig.3, where a microscope is plugged onto a RaspberryPI and controlled from a Taurus GUI that only uses a model provided by the *evaluation* scheme (actually it can be done by just passing the URI #8 from Table 1 as an argument of the "taurusform" launcher).

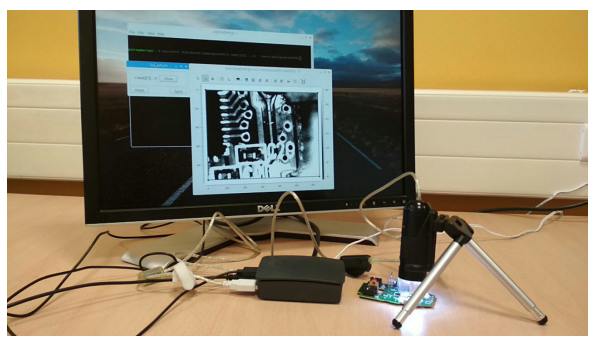

Figure 3: A "desktop lab" consisting on a microscope plugged onto a RaspberryPI. On the Screen, a Taurus GUI for controlling the connected equipment is shown.

CONCLUSIONS

We have shown that Taurus can be used to quickly deploy user interfaces in very different contexts. Its flexible design allows it to scale up and down to adapt to the requirements of both very large facilities and small laboratories.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

ACKNOWLEDGEMENT

We would like to thank the Taurus, Sardana and Tango community members and contributors. We would also like to thank Frederic Picca (Soleil) for packaging Taurus and Sardana for Debian.

REFERENCES

[1] C. Pascual-Izarra et al., in *Proc. ICALEPCS'15*, pp. 1138-1142.

[2] Taurus, http://www.taurus-scada.org

[3] PyQt, http://www.riverbankcomputing.com/software/pyqt

[4] Tango, http://www.tango-controls.org

[5] h5file scheme, https://github.com/taurus-org/h5file-scheme

[6] Sardana, http://www.sardana-controls.org

[7] T. Coutinho et al., in *Proc. ICALEPCS'11*, pp. 607-609.

[8] S. Rubio-Manrique et al., in *Proc. ICALEPCS'15*, pp. 1052-1055.

[9] S. Rubio-Manrique et al., in *Proc. ICALEPCS'11*, pp. 63-65.

[10] D. Fernandez-Carreiras et al., in *Proc. ICALEPCS'09*, pp. 709-711

[11] RaspBerry Pi website, https://www.raspberrypi.org