IMPROVING USER EXPERIENCE AND PERFORMANCE IN SARDANA AND TAURUS: A STATUS REPORT AND ROADMAP

Z. Reszela*, J. Aguilar, M. Caixal, G. Cuni, R. Homs-Puron, E. Morales,
M. Navarro, J. Ramos, S. Rubio, O. Vallcorba, ALBA-CELLS, Barcelona, Spain
M. T. Núñez Pardo de Vera, DESY, Hamburg, Germany
B. Bertrand, J. Foresberg, MAX IV Laboratory, Lund, Sweden
M. Piekarski, SOLARIS, Krakow, Poland
D. Schick, Max Born Institute, Berlin, Germany

Abstract

The Sardana Suite is an open-source scientific SCADA solution used in synchrotron radiation facilities such as ALBA, DESY, MAX IV, and SOLARIS as well as in laser laboratories such as the Max Born Institute. It is formed by Sardana and Taurus - both mature projects, driven by a community of users and developers for more than ten years. Sardana provides a low-level interface to the hardware, middle-level abstractions, and a sequence engine. Taurus is a library for developing graphical user interfaces. The Sardana Suite uses client-server architecture and is built on top of TANGO.

As a community, during the last few years, on the one hand, we were focusing on improving user experience, especially in terms of reliability and performance, and on the other hand renewing the dependency stack. The system is now more stable, easier to debug and recover from a failure. An important effort was put into profiling and improving the performance of Taurus applications during startup. The codebase has been migrated to Python 3 and the plotting widgets were rewritten with `pyqtgraph`. In addition, we also provide new features, like for example the long-awaited Sardana configuration tools and format based on YAML which is easy and intuitive to edit, browse, and track historical changes.

Now we conclude this phase in the projects' lifetimes and are preparing for new challenging requirements in the area of continuous scans like higher data throughput and more complex synchronization configurations. Here we present the status report and the future roadmap.

SARDANA SUITE OVERVIEW

Sardana is a Python-based scientific SCADA suite. Its primary goal is to reduce the cost and time associated with the design, development, and support of control and data acquisition systems. The suite consists of two independent projects: Taurus [1, 2] and Sardana [3, 4], and is built on top of the TANGO control system [5].

Taurus is a framework designed for creating user interfaces, including GUIs and command-line interfaces, to interact with scientific and industrial control systems, as well as other relevant data sources. GUIs are developed using PyQt.

Sardana, on the other hand, is a framework focused on the automation of experimental procedures and the control of laboratory equipment. It includes a powerful sequencer known as `MacroServer`, which incorporates a versatile scan and data storage mechanism. Additionally, it features a `Device Pool` that defines generic interfaces for laboratory elements and implements the hardware access layer. `MacroServer` as the client of `Device Pool` use Taurus core to implement the client side part of the communication. `Spock` which is based on IPython serves as a centralized command-line interface application for Sardana users.

USER EXPERIENCE

During the last years we have significantly improved the Sardana Suite user's experience with the most relevant changes listed below:

User Interfaces

Trends with Live and Archived Data The taurus_pyqtgraph trend widget [6], introduced the archiving support in the release 0.6. This development has fundamentally transformed the way users access and interact with historical data within the widget, offering valuable context for data analysis on a single tool, while using HDB++ as a source for archived information.

We achieved this by introducing two options:

- Loading Archived Data Once: Users can load historical data once and visualize it based on the date time axis they are exploring, this while the trend is continuously updated with new data being added.
- Automatic Data Loading: This feature allows users to navigate the date time axis retrieving and displaying archived data relevant to their current view with no need to click any button. This feature ensures that users maintain an up-to-date and continuous visualization of historical data as they explore different time frames.

Experiment Configuration The experiments can be configured using client applications, either the `expconf` GUI tool or directly through the `Spock` CLI. To ensure consistency, any changes made in one of the clients must be immediately reflected in all others.

The server notifies the clients about configuration changes with TANGO attribute change events, then it is the client's choice on how to reflect the incoming changes. Initially, this was solved by using *pop-up* dialogs that offered `expconf` user options to accept or discard the changes. However,

* zreszela@cells.es

in order to eliminate the inconvenience of these pop-up dialogs, the auto-apply option was introduced. Nevertheless, this option came with the drawback of exposing `expconf` users to the risk of data loss if someone else modified the configuration simultaneously from another client.

Finally, with the release of Sardana 3.3, the `view` and `edit` modes were introduced for using the `expconf` tool. Now, pop-up dialogs are only displayed when the tool is in `edit` mode, significantly improving the user experience and mitigating the risk of data loss.

Command Line Interfaces Multiple Taurus and Sardana console scripts have been consolidated into one script with sub-commands, similar to git. For instance, instead of `taurusform` or `taurusdemo`, we now have `taurus form` and `taurus demo`. Likewise, instead of `spock` and `macroexecutor`, we now use `sardana spock` and `sardana macroexecutor`. This change has resulted in a clearer and more accessible central point of access to all Sardana and Taurus-related tools. Additionally, this modification has led to a less cluttered binary namespace and has simplified packaging and documentation.

Furthermore, some of the Sardana macros have been modified to enhance user-friendliness. For instance, the macros for limiting motor positions have been renamed to be more self-descriptive. The motor position calibration macros have been improved to include the ability to recalculate software limits, and continuous scan macros now have an option to skip the last scan acquisition, making it easier to compare bidirectional scans.

Performance and Scalability

Taurus has performance issues affecting the startup time of certain types of applications. To improve the startup time performance we have created the *Taurus Performance Optimization (TPO)* [7] project. In the project we created baseline tests to find what code was responsible for slow starts of the user interface:

- TaurusForm: A PyQt application composed of a single TaurusForm populated with TaurusValue widgets (TaurusValue is composed from four TaurusLabel widgets).
- TaurusLabel: Similar to a TaurusForm but with just one TaurusLabel widget per attribute.
- Taurus core: Equivalent to as TaurusLabel case but without PyQt layer.
- PyTango: Equivalent to Taurus core case but without Taurus layer.

The baseline test revealed that the Taurus core has a significant impact on startup performance. In the Taurus core performance profiling we have uncovered areas for improvement. First and most obvious were the creation of attributes which generates a timeout, delaying startup by 6 seconds. A bug in Taurus's polling design can exacerbate this issue by further 3 seconds for attributes without change events. The second area of improvement was in how Taurus subscribes to configuration events, and by changing this, some extra time could be avoided for applications which do not need them. Finally, `TangoAuthority` objects were created multiple times, instead of reusing the same object between attributes.

The last two issues are already solved. Fixing the multiple creation of `TangoAuthority` objects reduced startup time by $\approx$ 21 %. Moreover, by making an event subscription optional, startup time may be reduced a further $\approx$ 29 %.

Debugging and Recovery

The next Sardana release, planned for the end of 2023, will come with the `expstatus` widget, an intuitive and comprehensive tool designed to provide real-time insights into the status of elements participating in an experiment. It proves particularly valuable in cases where a macro becomes unresponsive, as it offers immediate diagnostic feedback, pinpointing the issues and the elements affected.

Moreover, it not only serves as an information conduit but also facilitates user interaction with the system. In its initial phase, it enables the user to interrupt a macro through the *stop* and *abort* functionalities, as well as the subsequent release of associated elements. In a following phase, the widget allows the users to address elements that may be in a status other than *Ready* (e.g. *Fault*) by recovering them using the `reconfig` functionality.

Reliability

Sardana and Taurus have significantly improved their reliability thanks to enhancements in the TANGO control system framework. Developers from all three projects collaborate closely. On one side, Sardana and Taurus developers provide detailed issue reports and assist with testing, while on the other side, TANGO developers contribute enhancements and fixes. The most notable issues that were recently solved include: (1) Subscribing and unsubscribing from events in concurrent programming scenarios and Python Garbage Collector hooks. (2) Device restarting using the admin device `DevRestart()` command, which is used in Sardana's `reconfig` functionality. (3) Device server init hook, which is necessary when using a single Sardana server architecture. In the future, there are plans to implement dynamic attributes at the device level in TANGO, which are essential for the device `Pool`.

DOCUMENTATION

Sardana documentation has recently undergone significant improvements. The user documentation now includes a "What's New?" section, which describes the new features introduced in each release in a user-friendly language. It also provides links to more detailed chapters in the documentation. Additionally, we have started incorporating video demos into the documentation to offer more user-friendly explanations of the new enhancements.

On the other hand, the developer documentation has been expanded to include a detailed overview of the system architecture. This is aimed at attracting new contributors to the project.

CONFIGURATION

The standard way to configure Sardana is "interactive", using the `Spock` CLI and a set of configuration macros. The system is built up incrementally, by adding controllers and elements and tweaking settings. While this is intuitive, it has some drawbacks. Large systems are hard to manage this way. Things tend to stay around even when they are not needed, because they are not trivial to add back later. Furthermore, there is no general way to find out why something was added, when or by whom. It's hard to get the "full picture". These factors contribute to a tendency for Sardana systems to become "messy" with time.

In release 3.4, we have introduced a new feature - the "config tool". It is inspired by tools like Ansible, which define an entire configuration in a "declarative" way. The config tool is centered around a YAML file format which describes an entire Sardana system. The format is designed to be human read/writable, easy to generate programmaticaly and friendly to version control (e.g. git). The tool is intended to coexist with the interactive way of configuration.

The tool can set up a new Sardana system from such a YAML file, as well as creating a file from an existing system. A subsequent manual change in the Sardana system can be incorporated into the file without loss of comments or ordering, and conversely an update to the file can be applied to the system, without recreating it from scratch. These features together allow "round-tripping" the configuration between file and control system. The tool can also show the difference between a system and a config file. Finally, it can check a config file for syntax and logic errors. All these operations are visualized on the Fig. 1.

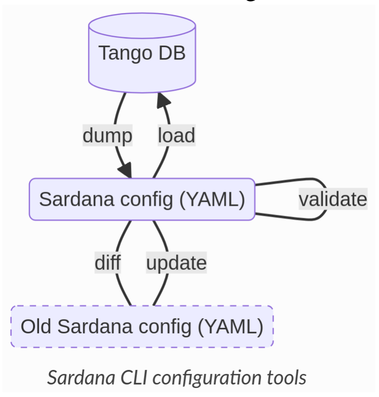

Figure 1: Sardana CLI configuration tools.

Two previous releases of Sardana brought two long-awaited improvements. Since release 3.2, the "interactive" method of deleting Device Pool elements has been protected, ensuring that an element cannot be deleted if another element depends on it. This ensures that the system always maintains a coherent state and starts correctly.

Both Taurus and Sardana provide global configuration options called "custom settings". Previously, the only way to change these configurations was by modifying the installed files (requiring write access to the installation directory). Taurus release 5.0 and Sardana release 3.3 introduced the configuration of "custom settings" using INI files. These INI files can be created at either the user-level or system-level, eliminating the need for write access to the installation directory. Additionally, this approach ensures that customizations are not lost during reinstallation.

DEVELOPERS EXPERIENCE

During the last years we have also significantly improved the Sardana Suite developer experience. Below you can find the most relevant changes:

Programming Interface

Taurus Mutlimodel Creation of a custom Taurus widget requires implementing a class that inherits from the `TaurusBaseComponent`. `TaurusBaseComponent` implements a model API (`setModel()`, `getModelObj()`, ...). Initially, this API allowed Taurus objects to be associated with a single `TaurusModel` object to act as their data source. The assumption of a single model worked for most cases, but it was too limited in the following two typical use cases:

- "model container" - refers to widgets/objects that can display an arbitrary number of models.
- "model composer" - differs from the containers in that it require a well-defined set of models, each providing the data for a specific aspect required by the widget.

As part of the TEP20 [8] the single model API was extended to support the above two use cases. The approach consist of supporting multi-models by assigning a "key" to each model, replacing the current attributes that imply a single model (e.g., `.modelObj`) with dictionaries that map those keys to the corresponding values, and adding an optional key keyword argument to the methods that access those attributes (e.g., `setModel()`).

Improved Trigger/Gate Controller interface To support highly customizable synchronization devices that enable the selection of input signals through configuration, the multiplexor mode has been adde to the Trigger/Gate Controller API. To further simplify Trigger/Gate controller development, the `SynchOne()` method now accepts configuration in the position domain using dial position units instead of user position units.

Type Annotations

Sardana is a large Python codebase, comprising over 100000 lines. It has been migrated from Python 2 to 3 as detailed below, but this conversion did not make further use of new features available in Python 3. Recently some effort has been made to add *type hints*.

Type hints are a way to add static typing to Python code. As Python is dynamically typed, the type hints are not used at runtime, but can be checked for consistency by tools such as `mypy`, and inspected by IDEs. Furthermore they provide documentation. More information can be found in *PEP 484* [9].

The approach for the initial conversion was to make use of the existing type information already encoded in "docstrings", documentation covering function argument types. Docstrings is a more informal way of type hinting, describing type information in text format. It has been used by the documentation generation system to build human readable API docs, and as such it was not very consistent. However with the help of some automatic tools (`doc484` and `com2ann`) plus application of `sed` and some manual work, it was possible to do the conversion in a few days. This provided type hinting for over 500 functions. While less than 10 % of the total number, it covers much of the core functionality.

Type hinting can be quite useful even if it doesn't cover the entire codebase. The initial goal is to be able to run static checks of the type consistency of major parts, and to start using type hinting during development of new code.

Testing and Continuous Integration

In Sardana 3.4 core `pytest` fixtures have been introduced, which allow easier, test-driven development of controller plugins. We identified `pytest` as the most complete and widespread python testing framework which provides "pythonic" and modular way of developing tests [10]. One can use the most common pool elements fixtures directly in tests (even without importing them as they are installed as `pytest` plugins) and eventually customize them using markers. An alterntaive is to create your own fixtures using the *factory as fixture* pattern.

One all-in-one docker image based on Debian 9 was used for CI testing. Supervisor was starting mysql, the pre-populated TANGO Databaseds as well as Sardana `Pool` and `MacroServer`. This image was rather large (3.6GB) and only one version of Python was tested (3.5). Thanks to some refactoring in the tests and the introduction of `pytest` fixtures to populate the database and start servers, testing locally or in CI became much easier: standalone mysql and TANGO Database docker images can be used. Those are started using GitLab services in CI and can be run locally using docker compose.

Those changes made creating new images to test Sardana more straightforward. They only need to include Sardana dependencies and we rely on conda-forge to install them. From one `Dockerfile`, we can create images based on `mambaorg/micromamba` [11] for Python 3.7 to 3.11. Adding a new Python version is simple. `micromamba` helps keeping the image smaller as it has no dependencies and avoids us having to use a two steps build. Those images are created in the sardana-docker [12] repository. Any change will trigger a downstream pipeline in the Sardana repository so we can ensure the new image isn't broken before releasing it.

To ensure the cleanliness and quality of Taurus code and to avoid the introduction of non-standardized code, two tools are utilized to aid and automate this process as part of the Taurus CI pipeline: `Black` and `Flake8`. `Black` ensures that the code adheres to the recommended PEP 8 formatting guidelines and automatically makes modifications to achieve this. `Flake8` not only checks the PEP 8 compliance but also highlights programming errors and assesses cyclomatic complexity, thus preventing the maintenance of low-quality code. Sardana will follow this recommendation and will incorporate these tools in the same manner as Taurus has done in the past.

Packaging

As seen previously, Sardana 3.4 introduced a new command line tool: `sardana config`. This subcommand isn't available by default after installing Sardana with pip. It was added to the PyPI package as an extra. Use `pip install sardana[config]` to install the optional dependencies needed. In Sardana 3.4.1, new extras were added: `spock`, `qt` and `all`. `pip install sardana` will install `pytango` but not `itango` anymore. Use `sardana[spock]` for that, which is equivalent to the `sardana` package from before 3.4. On conda-forge the Sardana package was also splitted in several subpackages: `sardana-core`, `sardana-qt`, and `sardana-config`. `sardana` is now a metapackage that will install everything (since 3.4). Those changes give more freedom to the users to install only what they need.

MAINTENANCE

Python 3

Due to the outdated software stack at ALBA [13], the Sardana and Taurus codebase had to remain compatible with older components like Python 2.6, TANGO 7, PyQt4, and so on for an extended duration. Even though the Sardana and Taurus community was well aware of Python 2 reaching its End of Life (EoL), the migration of the codebase to Python 3 consistently faced delays due to a preference for prioritizing other development efforts.

We first started putting efforts to adapt Taurus to Python 3, at the same time maintaining support to Python 2, using the `future`. This was achieved in Taurus release 4.5. It's important to note that Sardana, unlike the Taurus project, serves as a final application rather than a library. Consequently, it adopted a distinct migration strategy. The Sardana codebase underwent a transition to Python 3 without simultaneous support for Python 2 in its release 3.0.

At the time of writing of this paper all Sardana systems in all institutes of the community are already using Sardana 3.0 or higher. In case of Taurus there are still applications in different institutes using Taurus with Python 2, even thought the Taurus versions higher than 5.0, released at the end of 2021, removed Python 2 support after cleaning all the `future` module usage.

Migration of Taurus Plotting Widgets From PyQwt5 to Pyqtgraph

Originally Taurus implemented its plotting widgets, `TaurusTrend` and `TaurusPlot`, using PyQwt5. PyQwt5 has been unmaintained for a long time and not supported in most modern Linux distros and is not compatible with PyQt5 and Python3. As part of TEP17 [14] these widget were reimplemented using `pyqtgraph` library which is very efficient and offers a rich set of tools out of the box. The implementation strategy changed with respect to the previous one. First, the set of features added on top of the standard pyqtgraph widget are implemented as a composition of "tools" with the minimal interdependency between them, instead of the previous approach which included all the functionalities in one class. Second, the plotting widgets are maintained as a separate project, taurus_pyqtgraph, and are smootly integrated with Taurus by means of plugins. The scope of TEP17 covered only a basic subset of features that were supported in old widgets. The rest of functionalities, like statistics calculation, inspector mode or the possibility to change the curve titles, are being gradually added.

COMMUNITY

The Sardana community continues to thrive, with several active developers that coordinate their efforts through regular follow-up meetings. Some engaging events have played a crucial role in sustaining interest and involvement within the community. One of those is the Sardana Bug Squashing Party, which serves as a platform for introducing newcomers to the project while collaboratively fixing bugs.

Conversely, the Taurus community faced a period of reduced activity since its main developer left ALBA. In recent months, there has been a resurgence of activity, with several dedicated developers investing their time into the project. To foster this renewed engagement, the community has established regular follow-up meetings and initiated various events, such as a Taurus Workshop held in ESRF or a possible future Taurus Bug Squashing Party. These efforts reflect a renewed commitment to the ongoing development and growth of the Taurus project.

ROADMAP

The future needs of the Sardana community in the short and medium terms were discussed during a recent workshop held at SOLARIS [15]. The following actions were identified:

- **Sardana Configuration Tool**: Include `MacroServer` environment in the file format, support for multiple files and improve usage workflow.
- **Continuous Scans**: (1) Add multiple synchronization descriptions to deal with: *passive* elements e.g. shutter, sample environment, or *as fast as possible* acquisitions with elements reporting at different frequencies, (2) include a configuration tool for main-secondary relations between triggering elements, (3) publication of data to an in-memory database to decouple saving and reconstruction,
- **Macro API**: Redesign for a more flexible generation of custom scans.
- **Documentation**: Reorganize documentation to make it easier to follow and provide tutorials for users.
- **Reliability improvements**: Set user defined actions as scan recovery strategies. Address minor issues such as emergency brake and element states consistency.

The Taurus roadmap presents as following: (1) finish the performance optimization project; (2) ensure smooth adaptation of Taurus by the ESRF accelerators group; (3) ensure compatibility with PyQt6.

CONCLUSION

The Sardana Suite is a mature and production-ready solution for building SCADA applications in a scientific environment. Maintaining this software suite, which involves updating it with evolving dependencies, administering projects, and managing packaging pipelines, requires a significant amount of resources. Simultaneously, the continuous demand for more efficient and flexible software, which is common in the scientific environment, motivates us to continually enhance the user and developer experience while adding new features to Sardana and Taurus.

Thanks to its modularity, which is a notable feature of the Sardana Suite, adapting just a portion of it is as straightforward as adapting the entire suite. Recent decisions by the ESRF Accelerator Controls Group to adopt Taurus and the Max Born Institute in Berlin to adopt Sardana seem to confirm this.

Every software project faces the challenge of sustaining itself over time and adapting to evolving technologies. The strategy embraced by the Sardana Suite community is to foster collaboration, leverage our collective expertise, and share it with others.

ACKNOWLEDGEMENTS

We would like to thank the Taurus, Sardana and TANGO community members and, especially, to: F. Becheri, J. Gabadinho, A. Lopez (on leave), J. Moldes, A. Olle, S. Puso, C. Ramirez, X. Serra, S. Wohl (ALBA), J. Garrevoet, J. Kotański, T. Kraft, (DESY), G. Strangolino (Elettra), A. Hoffstadt (ESO), R. Bourtembourg, N. Leclercq (ESRF), M. Schneider (Max Born Institute, Berlin), A. Amhan (on leave), Á. Freitas, V. Hardion, A. Joubert, Y. Li, M. Lindberg, V. Silva (MAXIV), M. Celery, J. Kowalczyk, D. Trojanowska (MAX IV/S2Innovation), and M. Fałowski, M. Floras, A. Pytel, A. Switała, W. Wantuch, I. Zadworny, W. Zaręba (SOLARIS), Michał L. (on leave) (TANGO/S2Innovation) for their contributions to this work.

REFERENCES

[1] Taurus, `https://taurus-scada.org/`

[2] C. Pascual-Izarra *et al.*, "Effortless Creation of Control & Data Acquisition Graphical User Interfaces with Taurus", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1138–1142. `doi:10.18429/JACoW-ICALEPCS2015-THHC3O03`

[3] Sardana, `https://sardana-controls.org/`

[4] T. M. Coutinho *et al.*, "Sardana: The Software for Building SCADAS in Scientific Environments", in *Proc.*

ICALEPCS'11, Grenoble, France, Oct. 2011, paper WEAAUST01, pp. 607–609.

[5] Tango, https://tango-controls.org

[6] taurus_pyqtgraph, https://gitlab.com/taurus-org/taurus_pyqtgraph

[7] Taurus Performance Optimization repository, https://gitlab.com/taurus-org/TPO

[8] Taurus Enhancement Proposal 20, http://www.taurus-scada.org/tep/?TEP20.md

[9] PEP 484 – Type Hints, https://peps.python.org/pep-0484/

[10] pytest project, https://github.com/pytest-dev/pytest

[11] Micromamba, https://mamba.readthedocs.io/

[12] Docker images to test Sardana, https://gitlab.com/sardana-org/sardana-docker

[13] G. Cuní *et al.*, "ALBA Controls System Software Stack Upgrade", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 222–229. doi:10.18429/JACoW-ICALEPCS2021-MOPV037

[14] Taurus Enhancement Proposal 17, http://www.taurus-scada.org/tep/?TEP17.md

[15] Continuous Scans Workshop, 20th-21st September 2023, SOLARIS. https://indico.solaris.edu.pl/event/5/