ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA121

GENERIC DATA ACQUISITION INTERFACES AND PROCESSES IN SARDANA

Z. Reszela†, J. Andreu, G. Cuni, T. M. Coutinho, C. Falcon-Torres,
D. Fernandez-Carreiras, R. Homs-Puron, C. Pascual-Izarra, D. Roldan,
M. Rosanes-Siscart, ALBA-CELLS, Barcelona, Spain
M. T. Nunez Pardo de Vera, DESY, Hamburg, Germany
A. Milan-Otero, MAXIV Laboratory, Lund, Sweden
G. W. Kowalski, SOLARIS, Krakow, Poland

Abstract

Users visiting scientific installations aim to collect the best quality data frequently under time pressure. They look for complementary techniques at different sites and when they arrive to one they have limited time to understand the data acquisition architecture. In these conditions, the availability of generic and common interfaces to the experimental channels and measurements improve the user experience regarding the programming and configuration of the experiment. Here we present solutions to the data acquisition challenges provided by the Sardana scientific SCADA suite. In one experimental session the same detector may be employed in different modes *e.g.*, getting the data stream when aligning the sample or the stage, getting a single time/monitor controlled exposure and finally running the measurement process like a step or continuous scan. The complexity of the acquisition setup increases with the number of detectors being simultaneously used and even more depending on the applied synchronization. In this work we present recently enriched Sardana interfaces and optimized processes and conclude with the roadmap of further enhancements.

SARDANA INCREMENTAL AND ITERATIVE DEVELOPMENT

ALBA Synchrotron [1] was designed, constructed and commissioned in 2003 - 2012. In the selection process of its control system, commercial SCADAs competed against open source projects: EPICS [2] and Tango [3]. At that time these last two had been already successfully applied in numerous scientific installations. Finally the Tango Control System was selected [4] however its direct application to the experiment control was lacking many specific features like, for example, generic interfaces of the laboratory equipment, flexible sequencer or turn-key and customizable scans. Therefore, the Sardana project [5, 6], highly inspired on SPEC [7], was started in 2007. Its initial scope was mainly focused on the motion control, data acquisition and storage for the needs of step scans or similar measurement procedures. Works on the Taurus project [8, 9] dedicated to graphical user interfaces (GUI) started at a similar time. Both projects together compose the Sardana SCADA Suite which consists of Python libraries extendable with plugins and exported to the Tango control system.

† zreszela@cells.es

In 2013, DESY, MAXIV and SOLARIS together with ALBA established a collaboration on Sardana and Taurus. At the same time Taurus started gaining popularity within the Tango collaboration and became the *de facto* standard for building client applications in Python. From then on, many enhancements were proposed and implemented by the community of developers *e.g.*, native integration of diffractometers, hardware and software synchronized continuous scans or improved integration of one and two dimensional fast detectors (1D and 2D). In the next chapters we will concentrate on the data acquisition possibilities offered by Sardana.

STRIVE FOR OPTIMAL USER EXPERIENCE

When synchrotron users arrives to the laboratory they have limited time to understand the beamline components and the data acquisition architecture. At the same time they would like to tune the experiment parameters to achieve the best quality of data. Similarly, beamline staff newcomers would like to understand the data acquisition setup as quickly as possible to be able to exploit it to the maximum and prepare it in the best way for the upcoming users. For both use cases, a standard experiment control software is highly commendable.

The user interacts with the beamline control system viagraphical or command line interfaces (CLI). Each beamline at ALBA has, among others, one Taurus GUI application with a *synoptic view* of the line - a high level navigation tool enabling single-click access to the involved instruments. When interacting with the synoptic the *instrument panels* show all the relevant control and acquisition points represented by the *element widgets*. Thanks to the generic interfaces of the Sardana elements, any type of *moveable* (anything that can be controlled by commanding a set point *e.g.*, a motor, a temperature controller or a power supply) is represented by a *motor widget*. The same applies to the experimental channels: the *channel widget* provides a basic acquisition control interface to a counter, 2D detector, etc., hiding the complexity of the specific configuration behind an optional form with expert parameters. The live-view of the acquisition points *e.g.*, either motor's position or channel's current value, is as easy as drag-and-dropping from the element widget to plots or trends. The flexibility, immediacy and ease-of-use of this set of tools is specially welcome in highly dynamic contexts such as manual

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA121

alignment of a beamline or prototyping of new setups. Furthermore, Taurus user interfaces, can be built at minimal cost – not even programming but simply configuring few parameters, also in systems without Sardana or Tango [9, 10].

In Sardana, user experiments or other procedures are executed with the help of *macros*. A catalogue of predefined macros, including complex scans, exists. Custom macros can be easily programmed in the Python language.

The complexity of the experiment configuration comes from the numerous parameters offered to the user. First of all, the experimental channels and their corresponding synchronization must be selected. Also, in the case of the scans, it may be interesting to visualize their results online and hence the plotting needs to be configured too. Data storage options such as the data format or the destination path for the data file(s) also need to be configured. It is quite common to acquire a single snapshot of the beamline state just before starting the experiment and to store it together with the final data. All these parameters can be configured with the *experiment configuration widget* and can be stored persistently and reused between the experiments.

Spock is a complementary or alternative CLI application based on IPython [11], which can run stand-alone or be embedded in a GUI as another widget. It provides access to all elements of a given Sardana system and allows macro execution. Users not familiar with element or macro names may obtain help thanks to context-aware auto-completion and system-introspection macros. Experimental channels can be read from the CLI almost as easily as from the GUI (*e.g.* using the *count* macro to acquire over a given integration time and read the value of a channel or channel group). Also the same parameters that can be set via the experiment configuration widget, can be get and set using macros.

MEASUREMENT APPLICATIONS ON THE EXAMPLE OF SCANS

Numerous experimental techniques involve execution of scans. A scan is a measurement process that mainly consists of two actions: variation of a set point (one or more moveables) and data collection from detectors or sensors. Sardana offers two different modalities of scans: *step-based* scans and *continuous scans*. The former performs a sequential iteration of motion and acquisition stages, while the latter moves and acquires in parallel, significantly reducing the execution time. Also related, the *timescan* is a special type of scan where no moveable is involved and the data collection gets triggered based on preset time values.

Sardana continuous scans aim to be as similar as possible to their step equivalents. This is achieved on various levels: experiment configuration, macro input/output parameters and produced data [12]. The execution of continuous scans requires the synchronization of moveable elements, with detectors/sensors acquisitions. By default, best effort software synchronization is selected, but more precise results can be achieved by configuring and using hardware synchronizers. Three synchronization types are supported:

- Trigger - single acquisition is started by the synchronizer and automatically stops when the integration time elapses.
- Gate - single acquisition is started and stopped by the synchronizer.
- Start - the first acquisition is started by the synchronizer and the rest of them are governed by given integration and latency times.

The measurement preparation phase, explained in detail in the next chapter, enables optimal use of hardware with the capability to load a sequence of software-triggered acquisitions. This type of hardware pre-allocates the necessary resources, saving a significant amount of time in comparison to a preparation before each single acquisition point, and significantly reduce dead time of step scans (*e.g.* in the case of the ALBA Em# [13] the acquisition time was reduced by a factor of 10).

Data produced in a scan can be stored in arbitrary formats by means of recorder plugin classes. Sardana already comes with some built-in recorders like HDF5 [14] or SPEC. Each scan produces an entry in the output file, containing data from the moveables and the experimental channels. In the case of fast detectors, the processing and storage of their data by Sardana may be too inefficient. In this case, the data storage task is delegated to the detector itself or to an intermediate control software layer such as LImA [15] and then Sardana only stores a reference to the actual externally-saved data. Furthermore, if the HDF5 data recorder is used and the detector stores to HDF5 as well, then Virtual Data Set (VDS) references are created in the main output file, providing transparent access to all data from it.

Apart from the turn-key scan macros, custom step and continuous scans can be easily developed. A Sardana step scan is simply implemented as an iteration over a Python generator which yields the next moveables positions and acquisition integration times. By simply developing a custom generator the user can create new dynamic scans, for example, adapting to the results being acquired. A similar idea was reused in the continuous scans, where waypoint generators allow to compose complex measurement procedures from continuously scanned regions (moveables stop in between the regions), as demonstrated on the example of a continuous mesh scan. This last application, and many other examples of scans, could benefit from the support of non-linear motion trajectories currently not implemented in Sardana.

DATA ACQUISITION ON DIFFERENT LEVELS

Single Acquisition

Any experimental channel in Sardana can directly perform a *single acquisition* over a given integration time,

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA121

similarly as any moveable can be requested to go to a given set point. Single acquisition assumes basic configuration: software synchronization and timer acquisition mode. The user, whenever applicable, is still able to configure a timer channel proceeding from the same controller.

Grouped Acquisition

A *measurement group* is a high level Sardana element whose role is to coordinate a complex measurement process and to provide a simplified interface to it. It groups both the experimental channels and synchronizers and establish a relation between them.

The measurement process is divided in two phases: *preparation* and *data acquisition*. In the preparation phase, the measurement group configuration is loaded to the hardware. The configuration for the experimental channels whose acquisition is time-controlled by an internal or external hardware includes: the synchronization type (trigger, gate or start), number of repetitions (number of hardware triggers or gates to be armed), number of starts (number of software starts to be armed), integration and latency times and finally, the acquisition mode (*timer* or *monitor* and the corresponding channels playing this role). The synchronizers receive a synchronization description (described in detail in the "Trigger/Gate" section below). The measurement preparation phase, based on the element types and the configured synchronization, assigns the involved elements into actions: *synchronization*, *hardware*, *software*, *software start* and *0D*. The data acquisition phase starts the hardware and handles differently elements according to the action they belong to. The main activities during this phase are the continuous state interrogations (whether acquiring/synchronizing or done) and data (or data references) readouts which are subsequently passed to the downstream results collection.

Measurement groups usually involve multiple channels from the same hardware, (*e.g.* multiple channels of the same ADC card) which share the same communication interface with the hardware. In such cases, whenever the communication protocol allows that, grouped multi-channel commands and queries can be used in order to achieve better synchronization and to reduce the measurement dead time.

Autonomous Acquisition

Some hardware, apart from the capability to execute a controlled and synchronized acquisition, are able to perform an *autonomous acquisition e.g.,* continuous sampling of an ADC, a live video of a CCD camera, etc. Having two different control applications to deal with these two use cases introduces an extra complexity to the end user in terms of learning the application, memorizing different channel identifiers and configuring the channel. The challenge of combining both acquisitions in a single system lies in smoothly switching between them and correctly re-configuring the hardware. Sardana Enhancement Proposal (SEP) 17 [16] describes how to implement this new acquisition mode.

EXPERIMENTAL CHANNELS AND SYNCHRONIZERS

Sardana Suite is extendable with different types of plugins: *controllers*, *macros*, *data source schemes*, *recorders*, *widgets*. In order to foster the sharing of plugins within the community, the Sardana Plugins Catalogue [17] was recently created. Users are encouraged to search the catalogue for existing plugins before implementing their own, as well as to register their plugins in it to be reused by the community.

In this chapter we will focus on describing the details of the controller type of plugins, and more specifically, of the data acquisition related ones.

The controller plugins are used by Sardana to abstract and unify the access to specific hardware (or sometimes to software simulators or other control systems). They are implemented as Python classes following a defined interface which depends on the type of equipment to be controlled.

All experimental channels in Sardana have *value* and *value buffer* attributes: the former exposes the result of the most recent acquisition (maintains the cache with the result until the next acquisition is started) and the latter is a continuous acquisition buffer (with each value in the buffer identified so that an acquisition can be efficiently retrieved and reconstructed in blocks by a data collection aggregator). The common interface for experimental channels also defines *start* and *stop* commands to control the acquisition and a *state* attribute which indicates whether the channel is acquiring. The following is a description of the various types of data acquisition related controllers and interfaces currently supported by Sardana.

Timerable

Timerable controllers in Sardana represent hardware that allows time-controlled acquisition, *i.e.* that either has an internal clock allowing to pre-define acquisition intervals, or that it may be controlled by an external gate signal (including software start and stop commands). In the first case, the clock can serve to control acquisition of one or more experimental channels and is also represented as another channel in Sardana. In the second case, two pieces of hardware are chained together and one of them plays the role of the synchronizer. Current implementations of timerable controllers are the *counter/timers* and the *1D* and *2D channels*.

Counter/Timer is a timerable controller type used to represent hardware capable of counting digital signal edges or clock ticks. Sardana extends this concept to any hardware whose time-controlled acquisition yields a scalar value (admittedly, the counter/timer name is misleading but it is maintained for historical reasons). While a counter/timer acquisition is in progress, the hardware is periodically interrogated for the updated counter value and these is notified to the user.

1D and 2D are timerable controller classes representing channels that yield results in the form of one-dimensional and two-dimiensional arrays. In the last

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA121

decades detectors tend to produce larger volumes of data at a higher rate. Handling of such data in the controls software during the acquisition process may not be optimal and therefore the 1D and 2D controllers may either report the data or just a *reference* to it to the downstream results collection. In the case of working with references, the responsibility of handling the data (*e.g.,* storing or displaying it) can delegated either to the hardware, or to an intermediate control software, or to the controller itself. If a controller manifests this capability, the following extra attributes are added to the channel interface: *value reference* and *value reference buffer* (for accessing the references) and *value reference pattern* and *value reference enabled* (for configuring the data destination).

0D

Hardware without the possibility to time-control the acquisition but which can read scalar values may be integrated with the *0D controller* classes. During a Sardana acquisition, their channels are continuously read (software sampling) during the interval indicated by the user. The intermediate results are buffered and at the end of the acquisition, a custom function is applied to the buffer (*e.g.*, average or sum), and the result of such calculation is passed to the downstream results collection.

Pseudo Counter

A *pseudo counter controller* acts as an abstraction layer for one or more experimental channels (counter/timer, 0D, 1D or 2D) allowing the results to be transformed into a form meaningful to the user. Pseudo counters can be even build on top of other pseudo counters allowing more complex transformations while at the same time keeping the control software well structured and easier to follow. One example of a pseudo counter is *IoverI0*, which normalizes a measured channel result. New transformations can be integrated into Sardana with custom *pseudo counter* controller classes. The pseudo counter value gets updated automatically every time one of its experimental channel values get updated.

Trigger/Gate

Trigger/gate controllers represent synchronization devices such as digital trigger and/or gate generators. Their main role is to synchronize acquisition of the experimental channels. They are configured with a *synchronization description* composed by one or more groups, each group describing an equidistant sequence of gate signals parametrized by the *total* and *active* periods, the *initial delay* (or *initial position*) and the *number of repeats*. Non-equidistant synchronizations are possible by configuring multiple groups with different periods. The description is always expressed both in time and position domains (except for timescans, where the position domain is meaningless), allowing the coordination of time and position-aware synchronizers with a single configuration. In the time domain, elements are configured in time units (seconds) and the generation of the synchronization signals is based on the passing time. In the position domain, elements are configured in distance units referring to the "position" of a moveable element configured as the feedback source (this could be mm, mrad, degrees, etc.). In this case, the generation of the synchronization signals is based on receiving updates from the source. In summary, the trigger/gate controller class translates the synchronization description of a given measurement and configures specific hardware synchronizers.

ROADMAP

While the Sardana Suite is a stable product successfully applied in more than 30 laboratory setups, it is still under continuous development in order to catch up with the emerging software and hardware technologies.

In the nearest future we will focus on reducing the dead time in the step scans, finishing the implementation of the autonomous acquisition (SEP 17) and refactoring of the plugin systems in order to substitute the custom-made solutions with the current Python standards.

Next we will focus on exposing to the user an enriched unified interface of certain capabilities of the laboratory equipment regardless of the underneath implementation (hardware or software). Examples of those capabilities are regions-of-interest for 1D and 2D experimental channels, or limits enforcement and backlash correction for moveables.

Also, the support of non-linear motion trajectories could open new possibilities in relation with continuous scanning, and will be the mid-term milestone in the roadmap.

CONCLUSION

Sardana provides a gentle learning curve to the control system thanks to the consistent look-and-feel of its Taurus-based GUIs, its macro execution syntax familiar to SPEC users, and its python-based extendability. This improves the user experience especially for those visiting different beamlines of ALBA and other institutes within the Sardana and Taurus collaboration, as well as to those with previous SPEC experience.

Measurement processes in Sardana are composed from well isolated layers: macros, scan framework, measurement group, actions, controllers and elements connected by generic interfaces. This organization not only makes it easier to understand the data acquisition architecture of a given system, but it also promotes code reusability and facilitates the implementation of new features.

ACKNOWLEDGEMENTS

We would like to thank the Taurus, Sardana and Tango community members and the ALBA Computing Division and, especially, to Jan Kotański, Tim Schooft (DESY), Emmanuel Taurel (ESRF), Daniel Schick (Max Born Institute), Juliano Murari, Áureo Freitas, Antoine Dupré, Mirjam Lindberg (MAXIV) Stanisław Cabała, Michał Fałowski, Ireneusz Zadworny (SOLARIS) and Gabriel Jover, Jairo Moldes, Sergi Blanch, Sergi Rubio, Fulvio Becheri, Manuel Broseta (ALBA) for their contributions to

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA121

Sardana and Taurus. We would also like to thank Frédéric Picca (Soleil) for his help with packaging Sardana and Taurus for Debian.

REFERENCES

[1] ALBA, http://www.albasynchrotron.cat

[2] EPICS, https://epics-controls.org

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

[4] D. F. C. Fernandez-Carreiras *et al.*, "The Design of the Alba Control System: A Cost-Effective Distributed Hardware and Software Architecture.", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper FRBHMUST01, pp. 1318-1321.

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

[6] T. M. Coutinho *et al.*, "Sardana: The Software for Building SCADAS in Scientific Environments", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEAAUST01, pp. 607-609.

[7] SPEC, https://www.certif.com/content/spec/

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

[9] C. Pascual-Izarra *et al.*, "Effortless Creation of Control & Data Acquisition Graphical User Interfaces with Taurus", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 1138-1142, doi:10.18429/JACoW-ICALEPCS2015-THHC3O03

[10] C. Pascual-Izarra *et al.*, "Taurus Big & Small: From Particle Accelerators to Desktop Labs", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 166-169, doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

[11] IPython, https://ipython.org

[12] Z. Reszela, G. Cuni, C. M. Falcón Torres, D. Fernandez-Carreiras, C. Pascual-Izarra, and M. Rosanes Siscart, "Iterative Development of the Generic Continuous Scans in Sardana", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 524-528, doi:10.18429/JACoW-ICALEPCS2015-TUB3O02

[13] J. A. Avila-Abellan *et al.*, "Em# Electrometer Comes to Light", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 137-142, doi:10.18429/JACoW-ICALEPCS2017-TUAPL04

[14] HDF5, https://www.hdfgroup.org/solutions/hdf5

[15] A. Homs, L. Claustre, A. Kirov, E. Papillon, and S. Petitdemange, "LIMA: A Generic Library for High Throughput Image Acquisition", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper WEMAU011, pp. 676-679.

[16] SEP17, https://github.com/sardana-org/sardana/pull/478

[17] Sardana Plugins Catalogue, https://github.com/sardana-org/sardana-plugins