ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

DYNAMIC CONTROL SYSTEMS: ADVANTAGES AND CHALLENGES

S. Rubio-Manrique, G. Cuni, ALBA Synchrotron, Cerdanyola del Vallés, Spain

Abstract

The evolution of Software Control Systems introduced the usage of dynamically typed languages like Python or Ruby that helped Accelerator scientists to develop their own control procedures on top of the standard control system. This new high-level layer of scientist-developed code is prone to continuous change and no longer restricted to fixed types and data structures as low-level control systems used to be. This provides great advantages for scientists but also big challenges for the control engineers, that must integrate these dynamic developments into existing systems like user interfaces, archiving or alarms.

INTRODUCTION

ALBA [1], member of the TANGO Collaboration [2], is a third generation synchrotron light source in Barcelona, Europe. It provides synchrotron light since 2012 to users in its 8 beamlines, with 4 more in different stages of construction.

As a core member of the TANGO Control System Community [3], ALBA controls team has participated in the development of several tools and libraries shared across all institutes participating in the collaboration. Our main areas of development have been experiment control, GUI toolkits, alarm systems, archiving, simulation and dynamic user interfaces and device servers.

Control Systems in Modern Accelerators

Within the TANGO Community there are two clear different trends on developing accelerators control applications. Some of the institutes (ESRF, SOLEIL, Elettra, DESY) provide compact and uniform sets of applications to operators, applications mostly developed by control engineers. The newer institutes instead (ALBA, MaxIV, Solaris) have gradually taken a different path towards dynamically generated applications or user-developed interfaces on top of frameworks developed and deployed by the controls engineers [4].

This change has been gradual, as Java-based frameworks in TANGO already allowed some modularity and customizing of applications (ATKWidgets, JDraw), but the key factor has been the widespread usage of scripting languages (Matlab, Python) by accelerator scientists to write their own diagnostics and control software. An example of this practice is the Matlab Middle Layer [5] framework for accelerators control, that has become widely spread in the accelerators community, being used at ALBA to manage the Slow Orbit Feedback system [6].

Those scripting frameworks became a *de-facto* parallel control system and, although providing advantages to accelerator and beamlines scientists, but soon presented challenges in performance and unexpected behaviours in the control system, which required intervention from the Control team. These effects will be later explored in this paper, as well as the strategies used to cope with them.

Users as Developers

Despite the strategies that can be adopted from control and computing teams to keep up with control system changes, there's a fact that escapes from control teams and it's common for most scientific institutions: operators and scientists develop code on their own.

This fact has a practical explanation: most scientific careers include programming background, and the gap between programming and scientific languages is becoming smaller. Scientists feel enabled to translate to code their own ideas, and motivated scientists and long downtimes or operation shifts often lead to new toolkits or libraries developed by operators, scientists or users to enhance their daily work. These new tools often evolve from simple diagnostic tools to feedback control loops and GUI applications (Fig. 1), thus becoming at some point part of the control system. Sometimes it's a hard task for the Control engineers to adapt them to the integration/deployment workflows of the in-house controls team.

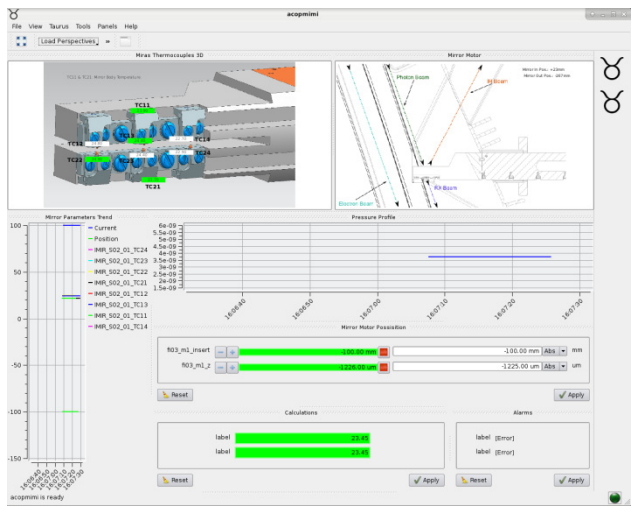

Figure 1: Taurus [4] GUI created by ALBA operators to manage the insertion/extraction of an infrared mirror.

To mitigate the cost of living with user-developed applications, three options can be adopted: forbid them to do it, give them total freedom, or adapt the Control System to their needs so they can continue developing but in a safer and more integrated way. Although the first option is the safest for the integrity of the Control System, it was not realistic nor acceptable for advanced users, so we started moving towards the third option, the design and development of a Dynamic Control System.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

DYNAMIC CONTROL SYSTEMS

The definition of what is a Control System it's highly dependent on the context, as the scope of that definition may change dramatically when comparing the systems engineering approach versus the computer science approach or what is finally understood in our scientific institutes. A simple definition would be that "a control system manages, commands, directs, or regulates the behaviour of other devices or systems". In a similar way, an SCADA, term that is also used together with Control System, would be defined as a "control system architecture that uses programmable logic controllers, computers, networks and graphical user interfaces for high-level process supervisory management".

As a scientific institution, our main "process" or activity from the user point of view is to provide the environment to perform experiments and tools for the acquisition and processing of data, but also to control several parallel processes (vacuum, radiofrequency, motion, cooling, protection, …) that will provide, enable or interfere directly or indirectly with experiment. But, experiments are not static processes, users may change daily (or every few hours), each with its different needs and targets of study. In the same way, our machines are subject to continuous upgrade, much more frequent that in most industry environments and many times subject to regression due to obsolescence or incompatibilities between the different elements to integrate.

So, let's say, that control systems in scientific institutions tend to be "dynamic" by definition. Dynamics, understood as "marked by usually continuous and productive activity or change", and thus requiring a continuous update to match experiment needs.

Control Adapted to Change

The first way in which we started adding dynamicity to our control systems is in the variability of attribute numbers for a given hardware. How many temperature readings we will require for a certain beamline? This number can be specified at design phase, but it will surely change with time as more diagnostic needs (or more elements) will appear. Thus, it is a reasonable assumption to consider that the number of temperatures to acquire will be variable.

Variability in attributes can be achieved in several ways: using arrays that vary in size (that may be a headache for clients or archiving), using by default an array much bigger than needed (thus a burden "just in case") or creating new attributes as they are needed (which implies some self-discovery or continuous update on clients).

Auto-generation

The previously described case of "dynamicity" is based on adding bigger numbers, or copies, of a kind of variable that already exist in our system.

The most challenging type of dynamicity appears when completely new elements appear in our system, or even when an existing element changes its interface, the way in which system variables are presented to the rest of the control system. These changes may be motivated for different reasons:

- A hardware upgrade, when same applications must manage newer devices that differ from the existing abstract class (e.g., IOT vs Solid-State RF plants).
- A software upgrade, when the communication strategy becomes different for a set of devices (moving from serial line devices to TCP-IP protocols, moving from a client-polling based system towards an event-based system).
- New ways to operate, accelerator technologies evolve, and we the user requires new functionalities that were not expected in the initial specification (e.g. new injection modes on a Linac).

One of the key advantages of the TANGO Control System is that it informs clients of the public and private interfaces of devices (Attributes, Commands and Property Lists, as well as its User/Expert level access) as well as of any change of it (via the InterfaceChange Event). This type of event may be triggered by devices whenever its interface changes on runtime, thus while executing the device control. This feature allows to generate and update device control panels on runtime (Fig. 2).

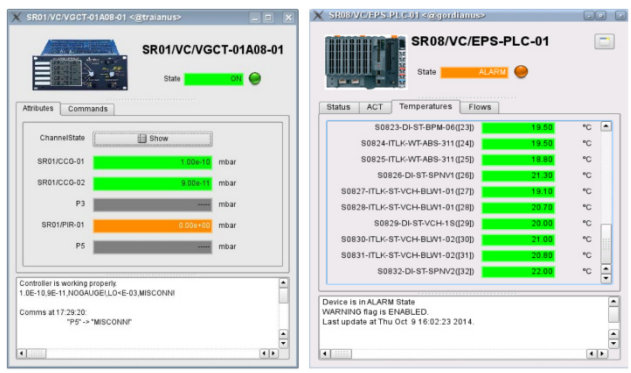

Figure 2: Different panels automatically generated for different types of devices in TANGO.

ADVANTAGES

User Scripts vs Dynamic Control Systems

Accelerator scientists (and physicists in general) had a long tradition of using mathematics-oriented programming languages like FORTRAN or MATLAB to develop their own calculation frameworks for machine designing. These frameworks have evolved to provide not only calculations and simulations, but real application in the development of control loops (as in ALBA's slow orbit feedback system, managed using Matlab Middle Layer framework and the TANGO-Matlab binding). But, the main disadvantage of these frameworks is that, as an external client not fully integrated in the Control System, they may not take into account its effects on the control

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

system in terms of cpu and memory usage, overall performance or long-term stability.

Providing a Dynamic Control System to the users can be seen as an opportunity to integrate these custom-made control processes into the standard control system, while still providing flexibility to scientist to develop their own code. At ALBA, we have managed to migrate many operation scripts from Matlab to Python, using our own Taurus [7] or Fandango [8] API's on top of PyTango [9]. This enables control engineers to encapsulate how user-made code is executed, and tune or limit the resources used by user applications to avoid effects on the performance of the standard Control System.

Composing and Prototyping on Runtime

This later case occurs when temporary installations or prototypes are introduced in the machine, and later on refactored or adapted to the definitive control infrastructure. As an example, at ALBA it occurred during the migration an upgrade of our RF Plants to solid-state amplifiers or new FPGA-based control loops. We had enabled different diagnostic devices to be editable by users and accelerator scientists, in a way in which scientists were able to modify those devices in order to match their different needs or the variables required by their mathematical models while also providing controls services like archiving or alarms.

These devices are divided into Façades, devices that provide a high-level interface over one or several hardware-related devices, or Composers, devices that summarize a large set of elements (like all the Ion Pumps in a machine sector) as one single element. Both kinds of devices provide attributes based on formulas (Fig. 3) that can be edited and updated by scientists on runtime without the need of a Control Engineer.

The same approach has been used in the development of simulator devices [10] for testing and prototyping purposes, as its versatility allows to duplicate almost any existing device in the Control System.

Device properties [test/sr/di]

Property name	Value
CheckDependencies	True
DynamicAttributes	CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N]) PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N]) Noise=sin(PI*t) Pressure=PRef+1e-7*Noise Current=CRef+2*Noise
DynamicStates	ALARM=Pressure>1e-8 MOVING=0<Current<20 ON=Current>=20 OFF=1
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
UseEvents	false

Figure 3: Declaration of Dynamic Attributes as properties in the Tango Database.

User-made Interfaces

The TANGO Database provides a repository with all Devices and Attributes available in the Control System, and the configuration parameters for its proper visualization. This centralized repository allows to develop GUI frameworks with capabilities to explore the database and allow users to build their own control tools by methods as simple as drag and drop. At ALBA it is achieved using the Taurus framework, that provides simple ways to create customized device panels and different views depending on user's context.

These user-made applications do not only provide direct access to devices data but also to the several services associated to the Control System. Access to Archiving [11] and Alarms [12, 13] is granted via python API's that allow to manage a complete control system from a single application [14] (Fig. 4).

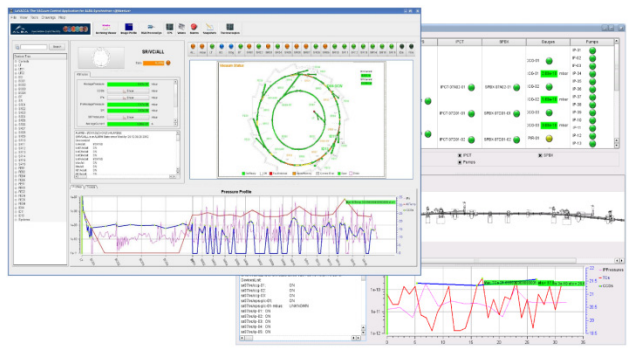

Figure 4: Vacuum Control Application, different perspectives with auto-generated panels.

Providing a framework to users to build their own applications has reduced the load on Control Engineers, as small changes in applications can be directly done by users to adapt to their own needs, including those changes that vary depending on the current mode of operation. Thanks to this versatility, only really complex or performance-demanding applications require now the involvement of a Controls Engineer. As technology and scientific demands continue to increase, Control Engineers workload has not been reduced, but can be dedicated to most demanding projects.

CHALLENGES

Cost and Consequences of Dynamicity

Continuous change and improvement in our institutes imply that the controls development team is periodically working on keeping up with the changes and upgrades in the installation. The tasks to keep the system up-to-date will include modifying graphical applications, tuning services like Archiving and Alarms and updating cabling databases and any other repositories of hardware information. New device classes must be integrated in the existing system, and it may trigger the need to adapt the existing abstract classes to include features that were not previously taken into account.

Unbalanced Load

A common solution for coping up with a changing system is developing applications and services that are capable of detecting and adding new devices as soon as they are added to the Control System database.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

This approach has many advantages and reduces the time needed to have devices available in applications when needed. But if the system is not able to scale properly, performance will be degraded gradually on time.

These effects are easily noticeable in slower applications, that become more and more bloated as the number of devices increase, and on services like Archiving that may get saturated once the number of inserted Attributes reaches the higher limits. The use of regular expressions or automated scripts to introduce elements in the Control System may be dangerous if all the tools do not take into account the increase in the service load.

Another kind of effects may appear when dynamically generated clients start saturating the hardware devices running below the Control System. The proliferation of "hidden" services (e.g., Alarms, Archiving, Composers, Façades) that are permanently accessing the controlled Devices could start to generate CPU usage problems if the system is not properly tuned. In the worst case, too many services accessing a Device will completely block it and impede any other client to access it.

Ghosts

Performance issues do not only occur when adding new elements into the system, but also when removing them. The effect of removing an element without properly updating databases and clients will immediately trigger problems in performance, as there's no worst task that searching for something that is not there anymore.

These performance issues may include timeouts on trying to establish connection, or long searches through the database in the case of not removing properly all the fields related to a device that is no longer present in the system.

Uncontrolled Code

Control Teams in most institutions have gradually adopted several strategies to maintain their code in a sustainable way. Continuous Integration and testing are widely spread, and Version Control is a must that no software team can avoid. But this is not the case of our software users. Although scientists develop code, they often do not know about common tools like SVN or Git, and sometimes even do not have access to the internal repositories of the institution. The code developed by Control System users may remain within operators' home directories, sometimes without regular backups or a regular control of changes from the last stable version.

This case is also valid for the code of generated applications or the formulas introduced in Dynamic Devices or the Alarm system; which are saved into databases and may not provide a history of changes if it is not introduced in the early design of the database (fortunately, TANGO did it).

The increase of user-developed code in our institution has gone in parallel to our demands to have better integration of testing in our development workflow. The difficulty to locate and verify the user code on each system upgrade is also a common cause of unexpected errors on software packages that were completely tested before delivery; sometimes because design updates or changes in API that were taken into account in Controls Team software couldn't be verified on users' code.

STRATEGIES TO KEEP UP

Python-based Control Systems

The development of a Dynamic Control System is enabled and shaped by the programming languages used on it. The TANGO core is written in C++, but supports an increasing number of bindings (Labview, Matlab, C, Python, Java). PyTango, the TANGO Python binding, is a wrapper developed on top of TANGO C++ Core libraries that provides all the functionality of TANGO and introduces multiple advantages on the development workflow.

Amongst other, some of the main advantages over other languages like C++ or Java are: dynamically typing (so more accessible to occasional programmers), most objects are mutable (can be loaded, modified and executed in runtime without compiling), and provides many already existing libraries and frameworks for managing scientific data (numpy, scipy, pymca, …).

PyTango also offers what is called the TANGO High Level API, an extremely simplified syntax for developing both TANGO clients and device servers. This API does not only reduce the required time to develop a device server, but also opened the possibility to write control applications to scientific users with certain programming background.

Today, most of the Device Servers and all Graphical User Interfaces (GUIs) at ALBA are developed in Python [15]. Most of them are not written directly over PyTango but using one of several python toolkits: Sardana (for experiment control), Taurus (for User Interfaces) and Fandango (for functional programming and dynamic device servers).

Dynamic Device Servers

Dynamic Attributes (variable numbers of attributes depending on device configuration) are one of the features that TANGO provides when developing a new Device Class. In addition, the Fandango library provides a template for adding new attributes depending on formulas written and executed on runtime.

These formula-based attributes, used by Composers and Façades and also in the PANIC Alarm System [16] devices, are often used to generate statistics, summaries or conditions depending on the combination of several Attributes and mathematical expressions. Attributes from different Devices or systems can be combined to create new results or states, that become available to all TANGO services and clients, even higher-level composers that may use this values to generate the global machine status.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

Applications Generated Automatically

Applications like PANIC, Taurus Archiving Browser or EPICS ChannelFinder are good examples of how a simple regular expression or search can be converted in a fully functional display of information. Based on the same approach, User Interfaces can be built automatically to summarize the information relative to all devices belonging to a same class (e.g., Power Supplies). This feature is enabled thanks to the TANGO Database, that allows to perform queries to extract all Devices matching a given name filter, class or host location.

This mechanism of auto-generation also allows to implement fast checks of the status of the system, or to verify its integrity detecting any change that do not match with the previously recorded state.

Dynamic Load Balancing

The first versions of Alarm [16] and Archiving [17] systems at ALBA were tied to specific subsystem configuration, what we call a dedicated service system. This approach has been kept for the oldest (and most critical) parts of the control system. But, for the newest parts of the control system we have adopted load balancing strategies.

To do so, we keep the dedicated system but not specifying directly attribute names, but attribute filters in the shape of regular expressions. It allowed to split the archiving of each subsystem in different databases while not enforcing to know the name of all attributes in advance.

In parallel, for each of the subsystem databases, the Archiving configuration API distributes the new attributes between the available data collectors based on the current load of each of them. Thus, providing the desired behaviour without compromising load balancing.

But, this mechanism of load balancing is highly dependent on enforcing a strict naming convention for the whole TANGO database. In the same way that we enabled a dynamic system it must be always accompanied with restrictions, to ensure that it will be consistent and maintainable in the medium term.

User Training and Version Control Enforcement

At ALBA, we have done several trainings for scientists and operators, to ensure that best practices are used and prevent future issues with code health, thus reducing the gap between control engineers and scientific users. Trainings have been done for both operators and scientists groups, either on python and git best practices or on how to use tools and libraries properly to not affect the overall system performance using ALBA version control repositories.

In the case of already existing or legacy user-code, we used two different options to track history:

- TANGO database, keeping history of last changes.
- filesystem-based version control, using snapshots provided by storage supplier, or open alternatives like gitfs [18].

Control System Profiling, Sandboxing

The dynamic behaviour of the system may cause some peaks in demand due to the elastic nature of the system, and the potential capacity of clients to affect devices performance. Although most institutes already deploy specific software to monitor memory, cpu and disk usage in their control hosts; some additional tools may help to detect and diagnose problems in the Control System.

At ALBA, we use the ProcessProfiler device server to track the most demanding processes in control hosts, providing the resulting statistics as TANGO attributes that can be archived or notified using the standard Archiving and Alarm systems.

TANGO already provides information regarding processes running on each host, the polling threads of each device and the times needed and available to read all required attributes. Each device also provides a Blackbox command to enumerate all the clients connected it and the type of operation requested.

But, as a best strategy to prevent performance degradation to be spread across the system, we separate critical and non critical devices in different systems. So, all protection systems are run by PLCs (static control system), critical Linux machines run on its own physical PC hardware and all dynamic devices are executed instead in dedicated virtual machines, ensuring that its performance do not affect any critical system managed by the same Linux host. We also export some control systems as read-only, using Composer devices as single-direction gateways. This is used with web applications (Fig. 5) to ensure that the system cannot be affected in any way by external client's performance or demands.

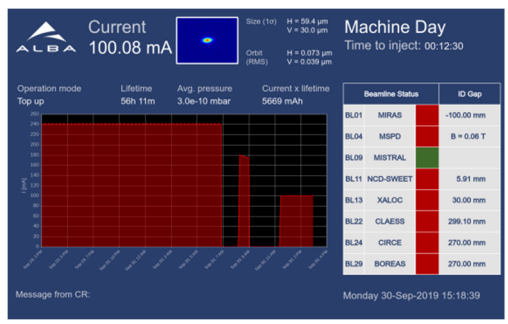

Figure 5: Machine Status Web Application [19], running on top of Composer devices running on virtual hosts.

In addition, we are migrating gradually our subsystems from a periodic polling approach towards event-based communications between devices and clients. Setting TANGO devices as the master allows to mitigate the possible effects of clients on hardware devices and reduces the required CPU usage. But this strategy should be accompanied with event filtering, as sudden event bursts that saturate the system.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOBPP05

CONCLUSIONS

This paper has explored the characteristics, advantages and challenges of introducing dynamic or user-programmed elements in control systems, as well as the different strategies that are being applied at ALBA to improve its usage.

The usage of user-tailored systems is going to increase gradually, becoming very attractive to users. This motivated that TANGO-managed scientific institutes that started operation or development in the last 10 years have adopted Python as its main programming language for both control and scientific developments.

REFERENCES

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

[2] R. Bourtembourg *et al.*, "Tango kernel development status", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 27-33. doi:10.18429/JACoW-ICALEPCS2017-MOBPL02

[3] TANGO, http://www.tango-controls.org

[4] C. Pascual-Izarra *et al.*, "Effortless creation of control and data acquisition graphical user interfaces with Taurus", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 1138-1142. doi:10.18429/JACoW-ICALEPCS2015-THHC3O03

[5] G. J. Portmann, W. J. Corbett, and A. Terebilo, "An accelerator control middle layer using Matlab", in *Proc. 21st Particle Accelerator Conf. (PAC'05)*, Knoxville, TN, USA, May 2005, paper FPAT077, pp. 4009-4011.

[6] J. Marcos and M. Munoz, "Slow orbit feedback and beam stability at ALBA", in *Proc. 4th Int. Particle Accelerator Conf. (IPAC'13)*, Shanghai, China, May 2013, paper WEPME038, pp. 3010-3012.

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

[8] Fandango, https://github.com/tango-controls/fandango

[9] S. Rubio-Manrique *et al.*, "Dynamic attributes and other functional flexibilities of PyTango", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, paper THP079, pp. 824-826.

[10] S. Rubio-Manrique *et al.*, "Reproduce anything, anywhere: A generic simulation suite for Tango control systems", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 280-284. doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

[11] L. L. Pivetta *et al.*, "New developments for the HDB++ Tango archiving system", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 801-805. doi:10.18429/JACoW-ICALEPCS2017-TUPHA166

[12] S. Rubio-Manrique *et al.*, "Extending Alarm Handling in Tango", in *Proc. 13th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11)*, Grenoble, France, Oct. 2011, paper MOMMU001, pp. 63-65.

[13] S. Rubio-Manrique *et al.*, "PANIC and the evolution of Tango alarm handlers", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 170-175. doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

[14] S. Rubio-Manrique *et al.*, "Unifying all Tango control services in a customizable graphical user interface", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 1052-1055. doi:10.18429/JACoW-ICALEPCS2015-WEPGF148

[15] D. Fernandez-Carreiras *et al.*, "Alba, A Tango based control system in Python", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, paper THP016, pp. 709-711.

[16] PANIC, https://github.com/tango-controls/panic

[17] S. Rubio-Manrique *et al.*, "Validation of a MySQL-based Archiving System for ALBA Synchrotron", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, paper WEP010, pp. 426-428.

[18] GITFS, https://github.com/presslabs/gitfs

[19] M. Broseta *et al.*, "A web-based report tool for Tango Control Systems via Websockets", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 826-829. doi:10.18429/JACoW-ICALEPCS2017-TUPHA173