ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

PANIC AND THE EVOLUTION OF TANGO ALARM HANDLERS

S.Rubio-Manrique, G.Cuní, D.Fernández-Carreiras, ALBA-CELLS Synchrotron, Barcelona, Spain
G.Scalamera, Elettra-Sincrotrone Trieste, Trieste, Italy

Abstract

The PANIC Alarm System is a python based suite to manage the configuration, triggering and acknowledge of alarms and automated actions in a Tango control system. The suite was developed at ALBA in 2007 and since then it has been adopted by several other facilities and installations such as Synchrotron light sources and large telescopes, integrating in the process a large set of community-requested features.

Its scalability is based on the stand-alone PyAlarm engines, that operate distributed across the control system; and the PANIC python API and user interfaces, that centralize the operation and configuration of the system. Each PyAlarm engine performs polled or event-triggered evaluation of alarm rules, complex logical operations and regular expression searches. The activation, recovery or reset of any alarm in the system can trigger actions like email, SMS, audible messages, local/remote logging, database insertion or execution of tango commands.

This paper describes the evolution of the suite, its compatibility with other alarm handlers in Tango, the current state-of-the-art features, the compliance with Alarm Management standards and the future needs.

INTRODUCTION

According to IEC 62682:2014 [1], the primary function of an Alarm System will be to notify abnormal process conditions or equipment malfunctions and support the operator response. The alarm system shall include mechanisms for communicating the alarm to operators via HMI or annunciators, as well as additional functions like event logs, alarm historian, automated actions and performance metrics.

Alarm Systems in Tango

The *Tango Control System* framework [2] is the result of a growing international collaboration to develop a modern open-source object-oriented control system that empower sharing control software development between institutions.

Three alarm systems are currently available in Tango:

- *Tango Alarm Handler*, by Elettra[3]: a centralized, event-based system evaluating syntax-rich formulas with high performance (C++), uses external devices as annunciators.
- *Alarm Archiving Database*, by Soleil Synchrotron: a centralized alarm logging database (Java), uses existing alarm configurations in Tango DB instead of formulas, records quality changes using polling.
- *PANIC*, by ALBA Synchrotron[4]: a distributed system that evaluates Python formulas, focused on flexibility has limited support for events, provides multiple annunciator types and automated actions.

Having several options is at the same time an advantage and an impediment if they are not complementary or able to interact. Thus, an effort to standardize the tools have been started based on the recommendations of the *IEC 62682 standard.*

The IEC 62682:2014 Standard

The *"Management of alarms sytems for the process industries"* standard (IEC 62682:2014) addresses the development, design, installation and management of alarm systems in the process industries. It defines the terminology and models to develop an alarm system and the work processes to effectively maintain the alarm system throughout the life cycle. It is based on ISA-18.2-2009 as well as previous documents reported by EEMUA [5] association.

As an standard, IEC 62682 provides a minimum set of rules and a guide for unifying criteria amongst countries/institutions. It provides recommendations on alarm systems design, implementation, and prevention of *alarm floods* (guarantee 12 < alarms/operator/hour). It also establishes that it is the responsibility of the operation group and the whole institution to ensure that all the procedures and stages of the standard are enforced, not relying only in the tools to do so.

THE PANIC ALARM SYSTEM

PANIC project was initiated during the construction phase (2007-2010) of the ALBA Synchrotron [6] to provide remote control of the several installed equipments on site (vacuum, Linac) while no operators were permanent on-site. In comparison with already existing alarm systems, PANIC is an scalable decentralized system that can aggregate hundreds of alarms from multiple Tango control systems or just run as a single process in an isolated IOC. Since 2013, PANIC has been adopted by several members of the Tango collaboration (Maxlab, Solaris, SKA).

PANIC Architecture

The PANIC Alarm System is based on two main entities: the *Alarm* object, keeping settings and state of each single alarm, and the *PyAlarm* device, a Tango device server on charge of evaluating a sub-set of alarms with a common configuration (on/off times, event count, annunciators setup, error management, ...).

PyAlarm device servers are deployed on the server side, loading each of them a list of alarms from an *AlarmAPI* collection (Fig. 1). Each PyAlarm device is an

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

independent process applying an specific setup to evaluate its formulas, export the results via dynamic attributes and summary arrays and perform logging and notifications via email, SMS or Tango commands.

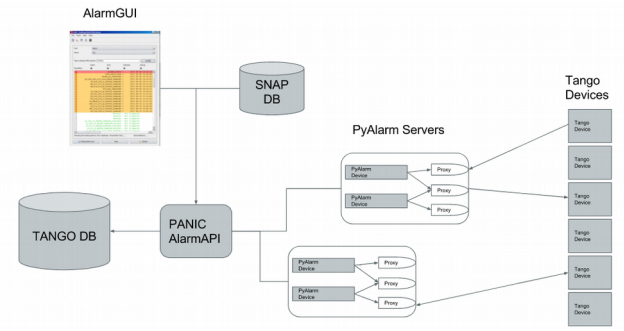

Figure 1: Panic Architecture.

The alarm system is entirely deployed by the *panic Python module* [7]. This library contains the classes for both the server and the client side, using the same API objects on both sides:

- panic.Alarm: Object that will keep both alarm configuration and state in servers and clients.
- panic.AlarmDS: Object that contains the configuration of an alarm evaluation engine and the methods to modify or manage it.
- panic.AlarmAPI: API object that allows to store/retrieve/manage alarm collections from a same Tango database origin.
- panic.AlarmView: API super-set that allows to sort and filter alarms from the same or different alarm systems, it can be used from both devices and clients.

The dynamic creation of a Tango attribute for each declared alarm allow any graphical user interface to display the alarm current value. This feature enables a single PyAlarm device server to be fully-featured as an Alarm System on its own.

In addition, the usage of unique tags for alarms allow to freely move alarms between PyAlarm servers that are managed by the same AlarmAPI. A feature that allows the operator to easily modify the timing of an alarm or the control engineer to manage the work balance between servers.

PyAlarm Tango Device Server

The PyAlarm device server performs polled evaluation of alarm formulas stored in the Tango Database, exporting its actual result, the attribute values used for the evaluation, and several methods for data inspection and alarm Reset/Acknowledge/Disable.

The key features that define the PyAlarm device are:

- single process capable to evaluate tango attributes
- notification (email/commands) included in the device
- minimum dependencies, just Tango and Python
- alarm state kept locally, logging local or remote

Each PyAlarm device is an evaluation engine with independent configurations regarding manage, events, logging, exception management, etc. Amongst others, these properties will be used to control the evaluation:

- PollingPeriod: period in seconds at which alarm formulas are evaluated.
- AlarmThreshold: number of successive positive results that will activate an alarm (multiplied by PollingPeriod provides the activation time).
- AutoReset: time period that controls the deactivation of an alarm once its condition is no longer active.
- Enabled: formula that will control the enable/disable for the whole device, it can be used for transitioning from maintenance to operation mode.

Many other properties are used to tune specific behaviours of the evaluation engine (events vs polling, cache depth, exception management). See the PyAlarm User Guide [8] for further reference. The HMI provides user validation to restrict the access to these properties.

Any Tango Device Server can interact with PANIC alarms using the PyAlarm device attributes and states as alarm triggers, or using their commands or attributes as annunciator actions. This same feature can be used to group alarms hierarchically as any alarm attribute can be reused in a higher level alarm formula.

While PyAlarm is the minimal an essential unit of the Alarm System, the PANIC project combines an ecosystem of tools [9] that can be used for IOC monitoring, logging, speech synthesizing and other types of annunciation including recovery actions over the control system (e.g. retire mirror in case of overheating).

Alarms Specification

Definition of Alarm objects in PANIC has been adapted to the terminology used in the IEC 62682 norm. Each alarm is defined by the following fields:

- tag: a **unique** identifier for this alarm within the current Tango Database, the API enforces that no other alarm could repeat an already used tag.
- formula: single-line code to evaluate, it allows to use any Tango attribute name as a variable to operate with, as well as regular expressions to operate on lists of values.
- device: the PyAlarm device on charge of evaluating the alarm formula and trigger its notifications.
- priority: unique set of categories for prioritization of alarms: ERROR, ALARM, WARNING, INFO, DEBUG, CONTROL
- message: a description of the alarm to be sent to the annunciators of the alarm, as well as links to the operators knowledge base and the procedures to follow.
- annunciators: list of email receivers, phone numbers or Tango commands or attributes with its arguments.

In addition to its defining properties, alarm current state is kept using different flags within the API object:

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

- state (enumeration)
- error (None or Exception message)
- disabled (timestamp)
- active (timestamp)
- acknowledged (timestamp)

In its initial releases Alarm states where just a combination of the boolean result of the formula and the quality (Alarm/Warning/Valid/Invalid) of the external attribute together with the values of its flags.

Although flexible, this approach required additional communication of values between clients and device servers. This has been improved increasing the number of alarm states (Fig. 2) and exporting a new json-like *AlarmSummary* attribute with the whole device status.

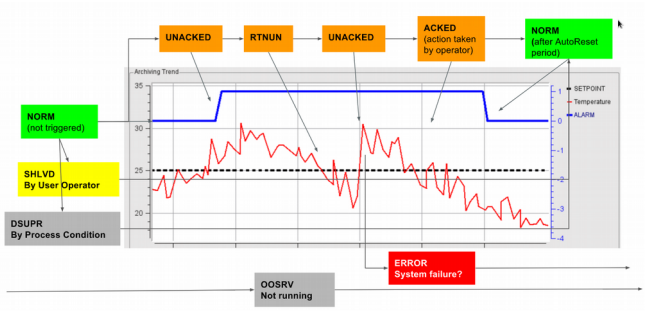

Figure 2: Alarm State Transitions.

The Alarm States has been redefined according to the IEC 62682 Standard. This conversion has been coordinated with Elettra's Tango Alarm System to unify the meaning of each alarm state within Tango. Alarms states and transitions can be seen in Fig. 2 and are governed by PyAlarm device properties and commands:

- **NORM** or Normal: Alarm evaluates to False.
- **UNACK** or Not Acknowledged: Alarm formula evaluated as True for a number of cycles above AlarmThreshold property, not yet acknowledged by operator.
- **ACKED**: Alarm is active but acknowledged by operator. Acknowledge/Renounce commands will manage the transition for UNACK and ACKED.
- **RTNUN** or Return To Normal Unacknowledged: Alarm condition is no longer True, but the alarm is still temporarily latched by its boolean attribute until a Reset() command is executed. This period is managed by the AutoReset device property.
- **SHLVD** or Shelved: Alarm evaluation has been temporarily disabled by operator using the Disable(timeout) command. Shelving will finish after timeout or PyAlarm restart.
- **DSUPR** or Design Suppressed: Alarm evaluation has been disabled by an evaluable condition, using the Enabled property. This condition affect to all alarms managed by the same device.
- **OOSRV** or Out Of Service: The PyAlarm device on charge of evaluating the alarm has been stop (intentionally).
- **ERROR**: The alarm formula cannot be evaluated, either by formula or attribute exception, timeouts or because of PyAlarm malfunctioning. This state is not included in the norm but useful to differentiate failed alarms.

Alarm Formulas Evaluation

Each PyAlarm device will evaluate the formulas of its assigned alarms at a fixed rate. The evaluation can be executed at a rate up to 10Hz, depending on the number of attributes and alarms to be evaluated. Attribute values are acquired by polling or events and cached to be reused in several formulas if necessary. The cache has a depth equal to the *AlarmThreshold* property, allowing to calculate alarms on deltas or averages of the last values.

Formulas, as shown in Table 1, are evaluated using the *TangoEval* object from *fandango* library[10], that provides pythonic[11] syntax and regexp searching for easily accessing Tango attribute values, qualities, timestamps, delta increments and received exceptions.

Attribute qualities can be used in PANIC to delegate the alarm condition evaluation to the hardware device, using internal set-points of the hardware instead or settings fixed by operators in the Tango database.

Table 1: Examples of Formulas Format (including regular expressions and wildcards, as seen in the docs[12])

T01_AL = bl01/plc/01/T01 > 30 or bl01/plc/01/T01.exception
T02_AL = bl01/plc/01/T02.quality in (ATTR_ALARM,ATTR_WARNING)
T03_AL = bl01/plc/T3.delta > 3
TEMPS_ALARM = T01_AL or T02_AL or T03_AL
TEMPS_ALARM = GROUP(T*_ALARM)
TEMPS_ALARM = any(t>30 for t in FIND(bl01/plc/T*))

PANIC extends TangoEval functionality with new methods like *GROUP*, to manage multiple alarms as one.

Alarm Annunciators and Logging

The primary function within the alarm system is to notify operators of abnormal process conditions or equipment malfunctions and support the response.

Each PyAlarm device is already capable to trigger multiple annunciator types for each alarm state transition: visual, SMS, email, Telegram and external commands. Each of these methods can be called generically just passing the email or phone number to the annunciators list or calling the PyAlarm commands (SendMail, SendSMS, SendTelegram) to generate customized messages.

This list can be extended using commands from other Tango devices, like FestivalDS that allows to trigger speech-synthesizing through control room speakers. Other Tango devices like SnapArchiving or FolderDS are currently used to enable remote logging either in database or JSON files. These external devices complement the log-

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

ging of alarms via local files for each PyAlarm or Tango Archiving [13] of alarm attributes.

Initially, annunciators others than email were only used whenever the alarm was transitioning from NORM to UNACK, but the ACTION keyword has been added to execute commands or attribute writing on Alarm, Acknowledge or Reset transitions.

As example of this behaviour, automatic aperture of ALBA front-ends has been configured using PANIC, to trigger the valves Open() command once all protection systems interlocks have been cleared.

HUMAN MACHINE INTERFACES

To help the operation and maintenance of any installation, the operators require alarm summary windows [14] to clearly identify any abnormal situation in the control system. The user interfaces must help the operators providing sorting, filtering and shelving functions as well as some guidance to the sequence of actions to be done by the operator (e.g. flowcharts).

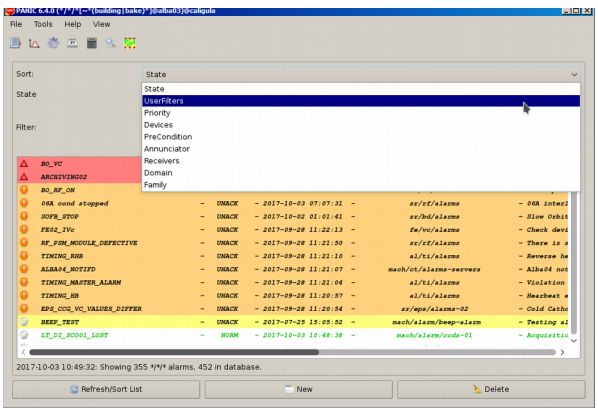

Figure 3: AlarmGUI widget, filtering options.

The PANIC AlarmGUI list widget (Figure 3) provides alarms ordered by state and the time of the first event that triggered the condition. Thus, first-to-come unacknowledged alarms will appear always at the top of the alarm list grouped by its state and priority. Once alarms are acknowledged the list will be reorganized to provide the most meaningful information in any case.

The AlarmGUI widget can be used either as an standalone application or embedded in other taurus-based applications like Vacca [15, 16].

Three applications have been developed separately for viewing alarm logs, either in files (FolderDS), Snap database or Elastic Search (Kibana); but each institute is currently using its own.

Rationalization, Classification and User Filters

One of the main objectives to achieve when designing an alarm system should be to prevent Alarm flooding [1, 14]. Any condition in which the number of alarms requiring operator action exceeds a frequency above 2 in a 10 minutes period will be considered hardly to manage, while any amount above 10 alarms in 10 minutes will be considered as an unacceptable alarm flooding. Classification and filtering of alarms must ensure that each operator only has to attend to alarms regarding its area, and prioritization will be used to avoid low-priority and non-urgent alarms to create unwanted noise.

In PANIC, Alarms can be classified using different criteria: state, priority, *DSUPR* condition, device name, device domain and family, annunciators and receivers. Once classified, the list can be filtered using extended regular expressions and saved as a *UserFilter* in the Tango Database. It enables each operation group (Vacuum, Diagnostics, Cooling, Radiofrequency) to customize its view of the application and hide those alarms that are not part of their interest.

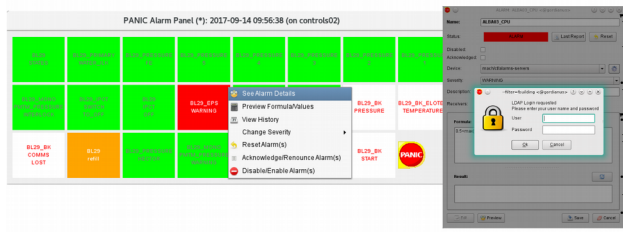

Figure 4: A UserFilter panel, allows pattern recognition to the operator and user-validated access to configuration.

Once Alarms are classified and filtered, a dedicated panel can be launched to display the alarms in which an specific operator is interested in (Fig. 4). This and other customized panels are built on top of the *AlarmView* API, that can aggregate data from multiple control systems and at the same time apply an stored *UserFilter* setup.

An example of a multi-host AlarmView is shown in Figure 5, where alarms from different beamlines are summarized in a single html table view.

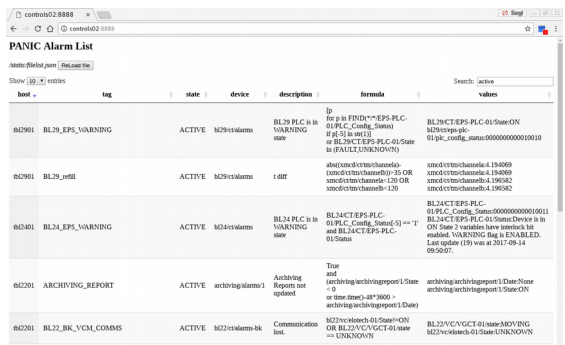

Figure 5: Web report [17] summarizing alarms from different beamlines.

User Validation and Action Logging

Permissions to modify Panic alarms are restricted for each user using the *TangoUserLogin* (included in PANIC) and *LdapUserLogin* (external) classes; that implement user identification using either encrypted hashes or LDAP credentials.

Users belonging to the list of *AdminUsers* will be allowed to modify any alarm in the system, while at the same time any user which email is receiving an alarm is

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

also capable to modify it. This gives flexibility to operators and technicians in shifts.

As required by the IEC 62682 standard, once User validation is enabled a local registry for all alarm changes is generated to keep track of any modification in the system configuration.

As permissions are stored in the Tango Database, they should be combined with the setup of the *TangoAccessControl* system to fully restrict unwanted access.

PANIC TESTING

A testing suite has been developed for PANIC in order to validate each PANIC release in continuous integration. The testing suite launches a SimulatorDS[18] device and 10 PyAlarm instances with different configuration setups. Each different setup can be easily configured using a csv file as seen in Table 2.

Table 2: Test Case Specification in csv Files

Device	Property	Value
test/panic/values	AlarmList	A_VALUE A_QUALITY
	A_VALUE .formula	test/panic/sim-01/T30>15
	A_QUALITY .formula	test/panic/sim-01/T30.quality == ATTR_ALARM
	AutoReset	0.0001
	PollingPeriod	0.5
	Alarm-Threshold	1

The obtained alarm values and its timing are compared against expected results via an additional PyAlarm device and the panic.ds.kpi library module, that extract metrics from the current Alarm system performance.

Key Process Indicators (like alarms/operator/10min and frequency log), will be added to the AlarmGUI widget in next releases, as required by the IEC 62682 standard for an alarm system in production.

CONCLUSIONS

The PANIC Alarm System has been already in production at ALBA Synchrotron for 10 years, a period in which it has grown and developed a large set of features. ALBA Accelerators and Beamlines are actually monitored by 123 PyAlarm devices, managing 657 formulas evaluating values from 2506 distinct attributes.

The convergence with Tango Alarm Handler and the adoption of the IEC 62682 have been the two final steps to convert it into one of the most complete open source solutions to implement alarm systems.

It is necessary to remark that software Alarm Systems are not yet an equivalent to plc-based protection systems. They fill a different gap in the control needs of a large facility as they are not part of the protection but of the response via human machine interfaces.

But, unlike other parts of the *HMI* that just translates the hardware into a console, the Alarm Strategy adopted by the operation group may change completely the final result and usability of the system. Despite of the tools used, the ultimate conformance with the standard is responsibility of the operation group, and they must ensure that a proper alarm philosophy and strategy is designed and applied.

Convergence of Tango Alarm Systems

Although PANIC supports Tango Events, the formula evaluation is not event-triggered as it is in the case of the Tango Alarm Handler device. This is a feature that has been requested several times and the current approach is to, instead of doing a complete redesign of PyAlarm, converge with the fully event-based Tango Alarm System and allow both systems to coexist and use the same applications and database schema.

This convergence required modifications in both systems that have been coordinated to achieve compatibility between both. These changes are summarized in [19]. Latest releases of both systems (September 2017) already apply these changes, making already possible to deploy hybrid C++/Python solutions that take maximum profit of its complementary features.

ACKNOWLEDGEMENT

We want to acknowledge the MaxIV development team and G.Jover from ALBA for its contributions to the PyAlarm Kibana and Telegram annunciators respectively.

Also the development teams at Solaris Synchrotron and SKA/TCS for its detailed debugging of PANIC against the latests releases of Tango and Taurus.

We also want to mention the work of the members of the ALBA operators group on refining the PANIC alarm system with its valuable experience.

REFERENCES

[1] "Management of alarms systems for the process industries", *IEC62682*, 2014

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

[3] L. Pivetta, "Development of the Tango Alarm System", *in Proc. ICALEPCS'05*, Geneva, Switzerland, Oct. 2005, paper WE3B.1-70

[4] S. Rubio-Manrique *et al.*, "Extending Alarm Handling in Tango", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper MOMMU001.

[5] *Engineering Equipment and Materials Users' Association* (EEMUA) issued publication, 191. University of Manchester

[6] D. Fernández *et al.*, "Alba, A Tango Based Control System in Python", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP016.

[7] The panic python module, https://github.com/tango-controls/panic

[8] PyAlarm User Guide, http://pythonhosted.org/panic/PyAlarmUserGuide.html

[9] S. Rubio-Manrique *et al.*, "PANIC, a Suite for Visualization, Logging and Notification of Incidents", *in Proc. PcaPAC'14*, Karlsruhe, Germany, Oct. 2014, paper FPO011.

[11] The fandango python module, https://github.com

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPL03

/tango-controls/fandango

[10] S. Rubio *et al.*, "Dynamic Attributes and Other Functional Flexibilities of PyTango", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP079.

[12] Fandango recipes, http://www.pythonhosted.org/panic/recipes.html

[13] L. Pivetta *et al.*, "New Developments for HDB++ TANGO Archiving System", presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUPHA166.

[14] Marcus Tennant, "Implementing Alarm Mangement Per the ANSI/ISA-18.2 Standard", *Control Engineering*, September 2013.

[15] S. Rubio *et al.*, "Unifying all Tango Services in a Customizable Graphical User Interface", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, paper WEPGF148.

[16] The Taurus Framework, http://www.taurus-scada.org

[17] M. Broseta, D. Roldan, S. Rubio, A. Burgos, and G. Cuni, "A Web-Based Report Tool for Tango Control Systems via Websockets", presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUPHA173.

[18] S.Rubio-Manrique *et al.*, "Reproduce Anything, Anywhere: A Generic Simulation Suite for Tango Control Systems", presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUDPL01.

[19] G. Scalamera, L.Pivetta, and S.Rubio-Manrique, "New developments for the Tango Alarm System", presented at ICALEPCS'17, Barcelona, Spain, Oct.2017, paper TUPHA165.