PANIC, A SUITE FOR VISUALIZATION, LOGGING AND NOTIFICATION OF INCIDENTS

S. Rubio-Manrique, F. Becheri, G. Cuní, D. Fernandez-Carreiras, C. Pascual-Izarra, Z. Reszela,
CELLS-ALBA Synchrotron, Barcelona, Spain

Abstract

PANIC is a suite of python applications focused on visualization, logging and notification of events occurring in ALBA Synchrotron Control System. Build on top of the PyAlarm Tango Device Server it provides an API and a set of graphic tools to visualize the status of the declared alarms, cre- ate new alarm processes and enable notification services like SMS, email, data recording, sound or execution of Tango commands. The user interface provides visual debugging of complex alarm behaviors, that can be declared using single-line python expressions. This article describes the architecture of the PANIC suite, the alarm declaration syntax and the integration of alarm widgets in Taurus user interfaces.

INTRODUCTION

ALBA[1], member of the Tango Collaboration[2], is a third generation Synchrotron lightsource in Barcelona, Europe. It provides synchrotron light since 2012 to users in its 7 beamlines, with 2 more under construction.

PANIC is an Alarm System running on top of the Tango Control System to provide periodic evaluation of user-specified alarm formulas, automatic actions and notification whenever formulas evaluate to True, logging of the control system status when this occurs and later monitoring and supervision of the evolution of the system.

Elements of the PANIC Alarm System have been deployed at ALBA[3] since the start of the construction phase. Developed to provide stand-alone monitoring during the vacuum installation of the accelerators it evolved into a versatile system in which many tools interact to provide not only a monitoring tool, but a supervisor service on top of a Tango Control System.

Other Alarm Systems existed already in Tango. PANIC was inspired on Elettra[4] (C++) and Soleil (Java) alarm systems; but focused on exploiting the versatility of python[5] to process rules on runtime, allowing operators and engineers to develop complex logics in rules[6].

THE PANIC ECOSYSTEM

The PANIC Alarm System is completely integrated in Tango[7]. Although it can work without a Tango Database using files as configuration, it develops its complete functionality when interacting in a complete Tango Control System.

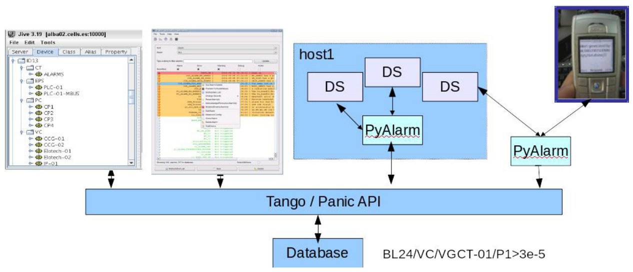

Figure 1: Architecture of the PANIC Alarm System.

The different elements that are part of a PANIC Alarm System (Fig. 1) are:

- PANIC Api: Alarms Database API and rule evaluator. Provides a unique view of the system and coherence between all devices and UI's. It encapsulates all the behavior that is later executed in the User interface or Device Servers.
- PyAlarm Device Server: Tango Device Server that, on top of the PANIC Api, executes alarm rules periodically and trigger the configured actions.
- ProcessProfiler Device Server, a Tango Device to inspect current performance of a linux system, exporting memory and cpu usage of all running processes to be used as source for Alarms.
- Festival Device Server: providing speech and pop-up notification in user terminals.
- PANIC User Interface: manager of the PyAlarm devices, editor of formulas and browser of the Tango Control System from alarms point of view.
- PANIC Tau Toolbar: simple Alarm viewer, restricted to alarms related to attributes shown in a running Tau application.
- Taurus Search Bar: it provides a search engine that indexes relationships between devices, attributes, properties, labels and alias within Tango.
- MySQL Databases: instead of having its own schema PANIC relies in databases of the Tango Control System for configuration (Tango DB) and logging (Tango Archiving DB).
- Snap Viewer: widget that browses the Tango Snapshoting Database (developed by Soleil Institute) where alarm logs and related attribute values are stored.
- Alarm NoSQL database: alternative database for unified configuration and logging, under development by Max IV team.

ISBN 978-3-95450-146-5

- Alarm Web Reports, static HTML pages generated by PyAlarm summarizing current status of all variables related to the alarm.

ALARM FORMULAS EVALUATION

Formulas are evaluated by the Panic API using the Fandango[8] library and the fandango.tango.TangoEval Class. This object is a singletone for each process and shares cache and all Attribute/Device proxy between all threads.

Alarm formula syntax accepts simple value comparisons, but also list comprehensions, regular expressions and aggregating existing alarms:

- MAX_P: `BL24/VC/VGCT-01/P1>3e-5`
- ALARM_P: `any([q in (ATTR_ALARM,) for q in FIND(BL/VC/VG-*/P*.quality)])`
- ANY_P: `MAX_P or ALARM_P`

Extended Syntax, Macros

Syntax available in formulas have been extended with some methods called Macros. Those methods provide a pre-parsing of the formula to execute in-place replacement before executing the formula.

Some examples of macros available:

- FIND(regex): that replaces regexp by all attribute names matching the regular expression.
- GROUP(regexp) : this allows to group many alarms as a single one, it returns a boolean flag if any matching alarm was activate in the last cycle.
- CACHE[attribute/alarm][-i]: returns access to the previous values of an attribute, keeping a buffer of size AlarmThreshold+1.

Extended Tango Attribute Name Syntax

Alarm formulas allow extended attribute names:

`some/device/name{/attribute}{.FIELD}`

Table 1: Examples of Using Extended Attribute Names

State (when no attribute is given)	`BL22/CT/EPS-PLC-01 == FAULT`
.value (optional)	`BL22/CT/EPS-PLC-01/CC1_AF.value > 1e-5`
.time (attribute not updated)	`BL22/CT/EPS-PLC-01/CPU_Status.time < (now-60)`
.quality (standard tango qualities)	`BL22/CT/EPS-PLC-01/OP_WBAT_OH01_01_TC11.quality == ATTR_ALARM`
.delta (e.g. for an open valve that just closed)	`BL22/CT/EPS-PLC-01/VALVE_11.delta == -1`
.exception (matched when the attribute is unreadable)	`BL22/CT/EPS-PLC-01/I_Dont_Exist.exception`
.all (full attribute struct with all previous fields embedded)	`[x.time<now-60 or x.exception for x in FIND(BL/PLC/01/*.all)]`

To provide full Tango functionality with a compact syntax the Alarms allow to express attribute names using extensions (Table 1). Those extra fields provide extra information from the device (even the type of exception when are unreadable) that enable complex alarms.

Those extra fields are also parsed whenever macros and regular expressions are used.

Triggering Automatic Actions from Panic

The syntax of Alarm receivers can be used to execute commands in outer devices, this kind of operations can trigger notification but also control actions. In the Table 2 example, a formula is used to automatically open a front-end whenever protection systems[9] allow it, and simultaneously notify beam-line scientists by pop-up.

Table 2: Example of Automated Front-End Opening Using PANIC. In this example an alarm triggers a PLC command and a pop-up notification in the beamline computers.

Formula	`bl/ct/plc-01/FE_AUTO and host:10000/chan/ct/fe/value and bl/ct/plc-01/BL_READY and not bl/ct/plc-01/fe_open and not bl/ct/-plc-01/fe_control_disabled`
Receiver 1	`ACTION(alarm:attribute,bl/ct/plc-01/OPEN_FE,1)`
Receiver 2	`ACTION(alarm:command:test/notif/blmachine/popup,$ALARM, $DESCRIPTION,15)`

PYALARM DEVICE SERVER

Each PyAlarm Tango Device contains a unique thread that evaluates a set of Alarm formulas written in python. Alarms are stored in the Tango Database as Device Properties: AlarmList for formula, AlarmReceivers for notifications, AlarmSeverities for categorization and AlarmDescriptions containing the message to be shown in notification and UI's.

The PyAlarm thread will keep a continuous polling thread (independent of Tango polling thread) evaluating the set of alarm rules and, depending on the device configuration, triggering notifications and actions when the alarm formula evaluates to True; and later managing Reminder, Reset, Acknowledge actions if the alarm state oscillates, recovers to False or is acknowledged by user.

Notifications by email and file logging are embedded in the PyAlarm device, while SMS, Speech and Popup require external device servers or plugins.

Distributing Alarms between Servers

A single PyAlarm device server can group many devices, but the threading configuration is unique to each device. Grouping the devices by targeted attributes allows to minimize attribute reading, as all formula evaluators share cache and the last value read per attribute. A reasonable balance should be kept to avoid massive

ISBN 978-3-95450-146-5

Tango clients with thousands of connections from the same machine.

PyAlarm Device Configuration

Parameters that are independent for each device are controlled by 13 properties:

- AlarmList: each Alarm is unique for the whole Tango Control System, the Panic API keeps consistency between all PyAlarm devices.
- Enabled, PollingPeriod, AlarmThreshold, AutoReset: All this integer properties will control the startup delay of the device, the frequency of alarm evaluation, the number of triggers required to activate the alarm and time after which the alarm will be auto-reset when the condition is not active anymore.
- Reminder, AlertOnRecover, FlagFile, LogFile, HtmlFolder: Those properties will control additional notification and logging durint the Activation/Reset cycle.
- CreateNewContexts, UseSnap: enable Alarm and attribute values logging in the Tango Snapshoting database. The list of attributes to be recorded for each alarm is automatically generated from the formula, but can be later modified by user.
- UseTaurus, UseProcess: Control how the access to attributes is done, either using Taurus to choose between polling and events or doing polling thread in a background process to optimize speed.

PANIC GUI APPLICATION

The Panic GUI (Fig. 2) shows the list of active or declared alarms. It provides several filters to search alarms: by state (active/inactive), activation time, severity, subsystem, receiver or historic values.

A text search is also provided that allow to locate alarms by any of the attributes used in formula or words used in description.

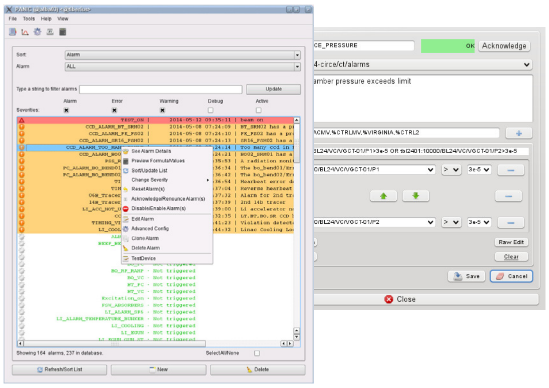

Figure 2: Alarm list and Alarm editor widgets.

For each alarm the menu allows to Reset or Disable the alarm, edit it, modify the PyAlarm device configuration, inspect the logging recorded in the Snapshoting database and access the "Alarm Calculator", a preview panel that allows to execute any formula using the Alarm syntax to preview the behaviour of the alarm when applied.

CONCLUSSION

The latest release of the PANIC Alarm System has evolved into a complete system that is becoming one of the main tool used by our operators and scientists. The strong pillar of the system is the reliability of PyAlarm, in operation for the last 6 years.

All the elements described in this paper are available in the Tango Device Servers repository in Sourceforge; thanks to the Tango collaborative effort now Panic is also used by other Synchrotrons like Max IV. The main focus in the next iterations will be further integration in Taurus and Sardana[10] to achieve certain usage uniformity between the several application involved.

It is still a fact that Tango has three different Alarm Systems instead of a unique tool. It is motivated by the three big families of developers (C++, Java, Python) being naturally attached to their preferred language. But, despite of having different rule-processing engines, actually some contacts have been made to increase the interaction between the three systems to allow notification services, graphical applications and databases to be combinable in the future.

ACKNOWLEDGEMENTS

The ALBA Accelerators division for the help in the debugging and refinement of the PANIC tools. A.Pearsson and A.Milán from Max IV institute in their effort to adapt and extend Panic as a tool for machine scientists.

REFERENCES

[1] ALBA website: http://www.cells.es
[2] TANGO website: http://www.tango-controls.org
[3] S.Rubio, et al. "Extending Alarm Handling in Tango", ICALEPCS'11, Grenoble, France (2011)
[4] Lorenzo Pivetta, "Development of the Tango Alarm System", ICALEPCS 2005, Geneva, Switzerland
[5] D.Fernández et al. "Alba, a Tango based Control System in Python", ICALEPCS'09, Kobe, Japan (2009)
[6] S.Rubio et al., "Dynamic Attributes and other functional flexibilities of PyTango", ICALEPCS'09, Kobe, Japan (2009)
[7] A.Götz, E.Taurel et al, "TANGO V8 – Another Turbo Charged Major Release", ICALEPCS'13, San Francisco, USA (2013)
[8] Fandango website: http://www.tango-controls.org/Documents/tools/fandango/fandango
[9] S.Rubio, et al. "PyPLC, A VERSATILE PLC-TO-PC PYTHON INTERFACE", PCaPAC'14, Karlsruhe, Germany (2014)
[10] T.Coutinho et al., "Sardana, The Software for Building SCADAS in Scientific Environments", ICALEPCS'11. Grenoble, France (2011)

ISBN 978-3-95450-146-5