EXTENDING ALARM HANDLING IN TANGO

S. Rubio-Manrique, F. Becheri, D.Fernandez-Carreiras, J.Klora, L.Krause*, A.Milán Otero, Z.Reszela, P.Skorek, CELLS-ALBA Synchrotron, Cerdanyola del Vallès, Spain

Abstract

This paper describes the alarm system developed at Alba Synchrotron, built on Tango Control System. It describes the tools used for configuration and visualization, its integration in user interfaces and its approach to alarm specification; either assigning discrete Alarm/Warning levels or allowing versatile logic rules in Python. This paper also covers the life cycle of the alarm (triggering, logging, notification, explanation and acknowledge) and the automatic control actions that can be triggered by the alarms.

INTRODUCTION

ALBA is the first Synchrotron Light Source built in Spain [1]. Its 3 GeV Storage Ring has been commissioned during 2011 and it's expecting the first beam-line users for beginning 2012. Control of ALBA [2] is based on Tango, a modern control system for scientific facilities developed by the ESRF and maintained by the members of the Tango Collaboration [3]. Tango provides a collection of interfaces for common devices and generic tools for configuration, deployment, archiving and alarms.

The installation and commissioning of ALBA required alarm handling to supervise accelerators conditioning and detect changes in operation conditions. Although two alarm logging systems already existed in Tango, at ALBA we decided to have an alarm system integrated in the existing applications instead of having a separated tool. To do so we combined previous ideas and developments to provide a more flexible alarm handler.

ALARM SYSTEMS IN TANGO

The Tango Alarm System

The Tango Alarm System [4] was developed at Elettra institute (Italy) by Graziano Scalamera and Lorenzo Pivetta. It uses a MySQL alarm database containing sets of rules that are permanently checked by a central daemon, the Tango Alarm Server, which logs alarm changes and triggers actions if needed. The rules are combinations of boolean operators and Tango Attribute values.

Table 1: Tango Alarms Syntax

sr/psch/s1.1/highthr
(({sr/psch/s1.1/stat} & 0x80) && ({sr/psch/s1.1/curr} > 15.0))

*On leave

The Alarm daemon is connected with the control system using CORBA Notification events [4] triggered by the targeted Attribute device; avoiding overhead in the system. The Alarm Server provides logging and is capable to launch commands of other Tango Devices, in which notifications and actions are delegated. All alarms in the system are stored and can be visualized using the graphical log-viewer tool.

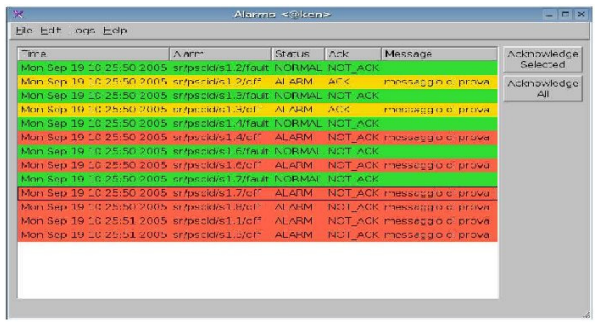

Figure 1: Tango Alarms Logs Viewer.

Soleil Alarm Database

The Alarm Database in operation at Soleil mimics the behaviour of Tango Archiving System [5][6], but it's focused on storing Attribute qualities (Valid, Invalid, Changing, Warning or Alarm) instead of values. The system uses a MySQL database and a pool of Archiver devices that poll periodically the quality of those Tango Attributes previously registered.

The conditions that trigger a change in Attribute Quality are not stored in the Alarm Database but in the Tango Database, using the Alarm/Warning Max/Min Values of each attribute. Configuration and visualization can be done using standard Tango Java tools and Archiving viewer.

ALBA PyAlarm

The PyAlarm device server development started on 2007 during ALBA's construction phase. It was focused on having an small stand-alone alarm system for installation activities (equipment tests, prototype labs, bakeouts, …) in places were database servers or network infrastructure was not fully available or could be interrupted.

Every PyAlarm Tango Device stores its rule sets as device properties, that can be stored in the Tango Database or Tango property files. These files allow the server to be running without Tango database if needed. Access between PyAlarm instances, clients and database have been encapsulated in the Panic API.

Alarm logging is done using Soleil Snapshoting database from the Tango Archiving System[5]. For every

Alarm triggered an Snapshot is recorded containing the Alarm Status and the value of every Attribute involved in the Alarm.

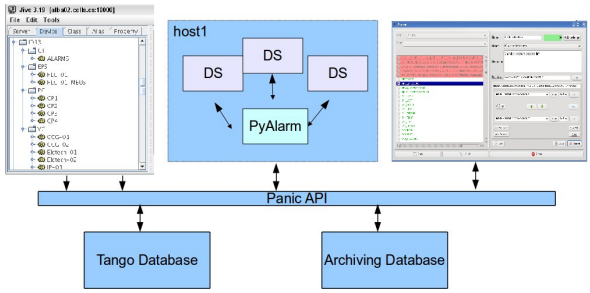

Figure 2: Pani c API enca psulates database access and provides alarm setup, validation and visualization.

A Python Alarm System

The PyAlarm rules are inspired in Elettra's rule-sets, with the aim of integrating both systems in the future. But PyAlarm applies python parsing; enabling a richer rule syntax with list comprehensions, regular expressions, string replacement and other functional features.

PyAlarm also creates dynamically [7] new boolean attributes for each new rule set, used to display alarm states from any generic Tango client. The attribute names syntax has been extended to combine conditions on value, quality and time-stamp of attributes.

PyAlarm uses polling when running on top of PyTango, but if Taurus[8] library is available the PyAlarm can use it to switch transparently between polling or events for each attribute.

Table 2: PyAlarm declarations showing syntax for value, host, state, quality and regular expressions.

```
BL_PRESSURE:
                        BL/VC/VGCT-01/P1 > 3e-5
BL_LOST:
              tbl01:10000/BL/CT/DB/State==UNKNOWN
BL_TEMPERATURE:
              BL/EPS/PLC/T1.quality == ATTR_ALARM
ID_TEMPERATURE:
           any(t>85 for t in FIND(ID/*/*/Temp*))
```

THE ALBA ALARM SYSTEM

Although every PyAlarm is an independent process that runs stand-alone, all the Alarm system is coordinated using the Panic python API. This software layer encapsulates the access between servers, clients and the Tango database. The API provides a way to access alarms configuration and modify existing alarm distribution; not allowing to have duplicated alarms in the system.

Once configured, alarm logging and notifications are managed independently by each PyAlarm device. Each PyAlarm device instance manages a collection of alarms distributed by domain/family. Archiving and configuration are centralized in our Tango and Archiving Databases.

Accelerators Alarms Architecture

In Alba Accelerators the PyAlarm instances are dedicated by subsystem (e.g. being Magnets and Vacuum alarm systems are independent processes) and this instances can be decentralized deploying every PyAlarm in the same industrial PC were the monitored system is running.

Alarms can be declared hierarchically to summarize a big amount of alarms in fewer notifications. Alarms can be used as variables within other alarms formulas, that will summarize the state of their primitives when generating reports.

Beam-lines Alarms Architecture

An independent alarm system is running on each of the beam-lines. As the number of devices and subsystems is much smaller it is not needed to distribute the alarm system and it is centralized in a few PyAlarm servers running in the same virtual server were the Tango Database is running.

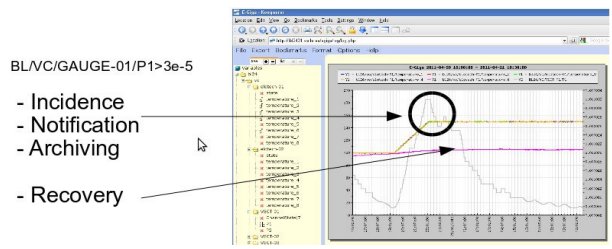

Figure 3: Messages sent during alarm life cycle.

The Alarm life-cycle

Alarms become active when the alarm condition evaluates to a True value, and will require human acknowledge to became inactive again. We added Reminder/Recovered notifications to avoid active and unacknowledged alarms remain unnoticed. Changes in alarm condition value will still trigger email and logging even if the alarm was still active, to make sure that no incidences remain hidden.

Table 3: Types of Messages

Alarm	The alarm condition has been activated.
Recovered	Alarm conditions are now inactive, but alarm state is kept.
Reminder	Condition was active for X period or became active after a Recovered period.
Acknowledge	Alarm has been acknowledged by operator.
Auto-reset	Alarm reset after being in Recovered state for a long time (optional).

Alarm Receivers

PyAlarm started as a notification service, so its main feature is email sending to notify any control incidence. It was initially extended to SMS and soon it was clear that more functionality should be needed. Every new notification feature has been added as a receiver type. In this sense an email address or a tango command are just different kind of receivers, that must be notified in case of incidence.

Every alarm has its own list of receivers, which may contain individual items or tags for groups of receivers. These groups are defined using email addresses, SMS numbers, lists of commands, archiving configurations or groups of them.

Table 4: Types of Receivers

email	Sent for every change in alarm status.
SMS	Sent only for activation.
Html files	To be loaded in the website.
SNAP	Record attribute values in the snapshoting database.
log	Generates a raw log file
COMM	Executes a TangoCommand.

USER INTERFACE

An easier method of global configuration/visualization of alarms helps accelerators and beamlines operators to diagnose incidences during commissioning; as well as modifying alarm conditions if needed.

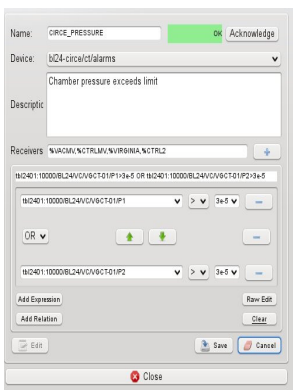

Figure 4: The Alarm editor widget.

An accelerator's alarm system requires an application for configuration, visualization and filtering of alarms usable at operator level; with no need of control system internals background.

The Alarm application allows to filter alarms by subsystem. To integrate alarms in existing applications has been developed an Alarm Toolbar with access to visualization and acknowledgement of alarms. This toolbar is able to filter alarms and status depending on user/application scope.

Alarms can be filtered by subsystem, receivers, attributes targeted and severity. Error, Warning, Info and Debug are the four severities available; which are used to sort the information when summarized in lists or toolbar. To go into detail as much as possible the alarm list provides viewer of attribute values.

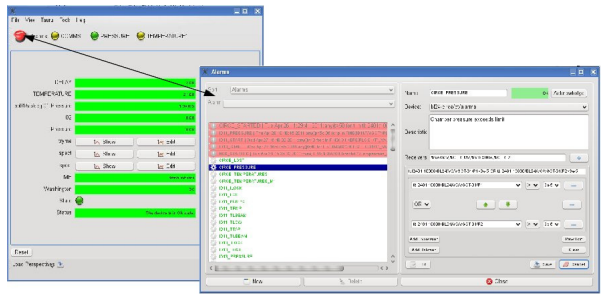

Figure 5: Alarm toolbar and list with filters and editor.

CONCLUSSIONS

We presented the ALBA Alarm System, created for ALBA installation and successfully extended to cover our accelerators and beam-lines. The PyAlarm was used successfully during installation and commissioning of ALBA linac, booster and beam-lines and in certain projects in the ESRF. The early deployment of an alarm system helped to prevent problems and detect irregular behaviours.

But for using it in our storage ring we had to improve the management of hundreds of alarms, adapting the content of messages and adding hierarchies between alarms that reduced the number of notifications sent. Our next objectives are the execution or recommendation of simple actions, linking alarms and low-level troubleshooting.

Unifying Tango alarm systems in a unique solution is still the aim of our development, so we focus next steps on interaction with existing systems; using PyAlarm as notification tool or adapting it to use Elettra Alarms Database as it exists now.

REFERENCES

[1] ALBA Paper D. Einfeld, “Progress of ALBA”, Proceedings of EPAC-2008, Genoa, Italy
[2] D.Fernández et al. “Alba, a Tango based Control System in Python”, ICALEPCS'09, Kobe, Japan.
[3] A.Götz, E.Taurel, J.L.Pons, P.Verdier, J.M.Chaize, J.Meyer, F.Poncet, G.Heunen, E.Götz, A.Buteau, N.Leclercq, M.Ounsy, “TANGO a CORBA based Control System”, ICALEPCS'03, Gyeongju, Korea
[4] Lorenzo Pivetta, “Development of the Tango Alarm System”, ICALEPCS 2005, Geneva, Switzerland
[5] E.Taurel, “Testing the Tango Archiving System”, Tango Meeting, 2004, ESRF, Grenoble, France
[6] S.Rubio et al, “Validation of a MySQL based archiving system for Alba Synchrotron”, ICALEPCS'09, Kobe, Japan
[7] S.Rubio et al., “Dynamic Attributes and other functional flexibilities of PyTango”, ICALEPCS 2009, Kobe, Japan