ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA165

# NEW DEVELOPMENTS FOR THE TANGO ALARM SYSTEM

G. Scalamera, L. Pivetta[1], Elettra-Sincrotrone Trieste, Trieste, Italy
S. Rubio-Manrique, ALBA-CELLS Synchrotron, Cerdanyola del Vallès, Barcelona, Spain
[1]also at SKA Organisation, Macclesfield, UK

*Abstract*

The TANGO Alarm System, based on an efficient event-driven, highly configurable rule-based engine named AlarmHandler, has undergone a deep refactoring. The dedicated MySQL database has been dropped; the TANGO database now stores all the configuration whereas the HDB++ historical database keeps all the alarms history. Correlating alarms with any other engineering data is now much simpler. A dynamic attribute is provided for each alarm rule; this allows to easily build a hierarchy of AlarmHandlers. The AlarmHandler manages Attribute quality in the alarm rules and provides possible exceptions resulting in alarm evaluation. Mathematical functions, such as sin, cos, pow, min, max and ternary conditionals are available in the alarm formulae. The TANGO AlarmHandler device server is now based on the IEC 62682 standard.

## INTRODUCTION

An alarm system is a tool that allows to notify operators of abnormal process condition or equipment malfunctions. The alarm system plays a central role in increasing uptime and overall service quality in the control systems of the Elettra and FERMI accelerators. In particular, the alarm system allows operators to be early notified of possible major faults, and to take the necessary action to prevent them.

Building on the experience matured during several years of use, with thousands of alarms deployed, a number of improvements and new functionalities for the TANGO alarm system have been designed and implemented.

## EVOLUTION OF THE TANGO ALARM SYSTEM

The first alarm system for TANGO has been developed at Elettra in 2004 [1]. The main requirements identified were:

- easily configurable at runtime
- support for complex alarm rules, with logical, binary, mathematical operators and functions
- support for alarm rules based on values gathered from multiple subsystems
- consistency of alarm states between multiple clients.

Named Alarm Collector, the TANGO device was designed to be completely asynchronous, exploiting the freshly added publish/subscribe support in TANGO. Every attribute involved in the formula is subscribed for the change event; when the Alarm Collector receives a new event, the evaluation of all the matching alarm formulae is triggered. The state of alarms was exposed to clients as an array of strings, containing all the necessary information: the actual condition of the alarm, the acknowledge flag, the timestamp, the severity, the group and the optional message associated to the alarm.

During 2008 the Alarm System undergo the first refactoring. GNU Flex and Bison, used as lexical scanner and parser, have been replaced with a novel parser developed with the Spirit Parser Framework [2], an object oriented recursive descendent parser generator implemented using template meta-programming techniques. As part of the Boost libraries, the Spirit Parser is written in C++, and use operator overloading to compose parser objects. The alarm parser has been implemented to build an Abstract Syntax Tree (AST) as the result of the parsing of the formula. Thus, each alarm rule is parsed only once at device initialization, and the evaluation is done using the AST, that results in better performance.

Furthermore, the capability to store the history of alarms, including the state and message, into a dedicated MySQL database has been added. The original schema included two tables: the '*description*' table, to store the configuration, and the '*alarms*' table, keeping track of the alarm history.

The possibility to execute a TANGO command as the alarm condition changes has been added. Two different commands can be configured, the first to be executed when the alarm condition changes from NORMAL to ACTIVE, the second when it changes from ACTIVE to NORMAL. The commands may have no input parameter or DevString input parameter. In the second case, the DevString is filled with the alarm name, group, message and the values that triggered the alarm condition using a 'key=value;' representation.

Also, the support to email notification triggered by alarm state change has been added. A new TANGO Device, named AlarmMail, has been developed in Python, providing mailing capabilities. This device provides a phonebook and a mailing list that can be associated to alarm groups. The commands SendAlarm and SendNormal, which accept a DevString argument, allow to send emails reporting an alarm or a normal notification. These commands can be configured as the actions to be triggered by the alarms to provide email notification.

## DESIGN GUIDELINES

A number of requirements has been taken into account during the design phase of the deep refactoring of the alarm system:

- drop the dedicated MySQL database used to store the configuration and the history
- save the alarm configuration in the TANGO DB
- save the alarm history using HDB++ [3]
- use the state of an alarm in the formula of another alarm
- comply to the IEC 62682:2014 standard

- extend the formula parser to support more mathematical and logical functions
- disable/enable alarms at runtime
- share, if possible, a common set of interfaces and functionalities with PyAlarm (PANIC's device server) [4], allowing to use the PANIC GUI.

## ALARMHANDLER

The new device server, core of the alarm system, has been named AlarmHandler.

### *Interface*

One of the key functions in the upgrade of the TANGO Alarm System is to create one dynamic attribute for each alarm rule. The attribute allows to asynchronously notify clients interested in alarm changes, with the granularity of one alarm. In fact, every attribute representing an alarm can push events when its state changes. Clients can subscribe for the events of the alarm attributes they are interested into, or all the alarm attributes, and be notified asynchronously. Clients can be graphical user interfaces, notifiers, other AlarmHandlers or any TANGO device in general.

To code useful information in the alarm attributes, the IEC 62682:2014 [5] standard has been adopted: the alarm attribute is a DevEnum TANGO data type and the possible values are NORM, UNACK, ACKED, RTNUN, SHLVD, DSUPR, OOSRV. Thus, the alarm attribute, in addition to the basic normal/abnormal process condition information, also carries the information whether the alarm is acknowledged and enabled. Furthermore, the quality of the attribute can be consistent with the quality of attributes involved in the formula, and an exception can be thrown in case the formula cannot be evaluated. Exceptions, in fact, can occur for multiple reasons, being generated by the TANGO attributes in the formula or by the parser.

To support shelved (SHLVD) and out-of-service (OOSRV) alarm values, some commands have been added to the AlarmHandler device. The *Disable* command permanently disables an alarm, until the *Enable* command is issued. The *Shelve* command temporarily disables an alarm, and its evaluation and notification, until the configured shelve time expires or *Enable* command is issued.

An additional attribute, named *alarmSummary*, has been implemented containing all the alarms information represented as an array of strings, one string for each alarm configured in the AlarmHandler. The strings are formatted using a 'key=value;' syntax, as in the following example:

*'tag=alm_name;state=NORM;priority=FAULT;time=2017-10-10 16:45:00.000;formula=(a/b/c/d==ON); message=msg'.*

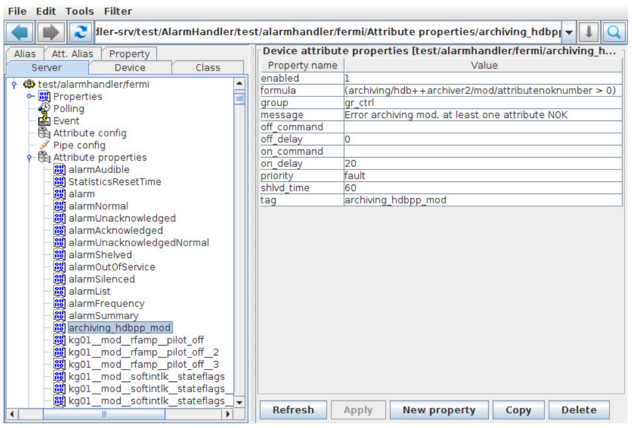
This allows graphical user interfaces interested in managing all the alarms configured in one or more AlarmHandlers to easily retrieve the whole bunch of information.

Moreover, to retrieve the detailed state information of a single alarm, the command *GetAlarmInfo* has been implemented. Additional information is returned, such as attribute quality, alarm counters, groups and attribute values. Attribute values, in particular, contain all the attribute values, or exceptions, resulting from the last evaluation of the formula, coded as a JSON string. A replacement of the *alarmSummary* attribute and *GetAlarmInfo* command with an implementation based on a TANGO DevPipe, introduced since Tango 9, is under evaluation.

### *Alarm Configuration and Alarm History*

The original TANGO alarm system used a dedicated MySQL database to store alarm configuration and alarm history. This additional component can possibly be avoided, since the configuration can be stored into the TANGO database, and, with the new dynamic attribute design, the alarm history into the TANGO historical database.

Since the new design provides one attribute for each alarm, Attribute Properties are the best place to store the configuration: for every alarm attribute, the properties to store the formula, the message and all the other optional parameters can be easily created and managed with Jive, as shown in Fig. 1, or programmatically.



Device attribute properties [test/alarmhandler/fermi/archiving_h...

| Property name | Value |
|---|---|
| enabled | 1 |
| formula | (archiving/hdb++archiver2/mod/attributenoknumber > 0) |
| group | gr_ctrl |
| message | Error archiving mod, at least one attribute NOK |
| off_command | |
| off_delay | 0 |
| on_command | |
| on_delay | 20 |
| priority | fault |
| shlvd_time | 60 |
| tag | archiving_hdbpp_mod |

Figure 1: Configuration in attribute properties.

In the AlarmHandler device, the alarm dynamic attributes are configured to push events every time the attribute value changes; they can be easily, and suitably, archived using HDB++, taking advantage of the existing tool. Thus, the alarms history can be easily correlated with any other data archived with HDB++. Figure 2 shows the correlation between the state of an alarm and the value of one attribute used in the alarm formula. The specific configuration is the one shown in Fig. 1 for the alarm *archiving_hdbpp_mod*, which requires *archiving/hdb++archiver2/mod/attributeNokNumber* to be greater than zero for more than 20 seconds (on_delay property). Shelved time is 60 seconds.In particular, the following 4 situations can be seen in Fig. 2:
(1) After 20 seconds of abnormal process condition, the alarm state changes to UNACK, then the Acknowledge command is issued; then the process condition returns to normal.
(2) After 20 seconds of abnormal process condition the alarm state changes to UNACK, then when the process

condition returns to normal the alarm state changes to RTNUN, and after the Acknowledge command to NORM.
(3) After the Shelve command the status of the process is ignored; after 60 seconds (shlvd_time property) the state returns to NORM.
(4) After the Disable command the status of the process condition is ignored, until the Enable command is issued.

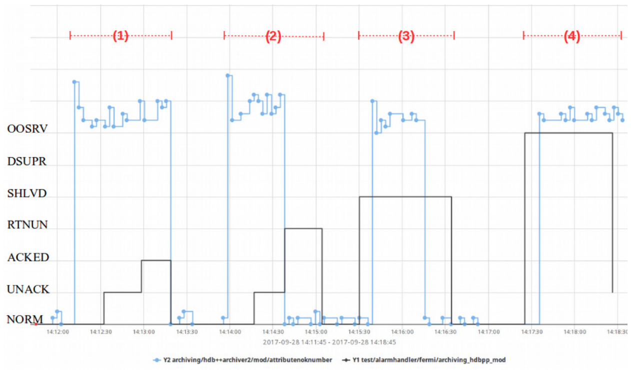


Figure 2: HDB++ history plot of one attribute (blue line) and of the alarm depending on its value (black line). It is possible to see the effect of the *Ack*, *Shelve* and *Disable* commands in different conditions.

## *Alarm Formula*

Some work has been done to extend the parser and support more predicates. The mathematical functions ‘sin’, ‘cos’, ‘pow’, ‘min’, ‘max’ are now supported. Adding additional mathematical functions is easy and can be done whenever a new requirement arise.

The support for ternary conditionals has been implemented, with the following syntax: *(expr1 ? expr2 : expr3).* If *expr1* evaluates to true than *expr2* is evaluated, otherwise *expr3* is evaluated.

Also, support for the quality of attributes has been added to the parser. Now it is possible to use the syntax *name/of/device/attr.quality* in formulae, and it evaluates as the integer number of the DevEnum quality. Furthermore, the syntax ‘*quality(expression involving attributes)*’ is allowed and evaluates as the combination of the quality of attributes in the expression. The result will be:

- *INVALID* if at least one attribute is invalid
- otherwise, *ALARM* if at least one attribute is in alarm
- otherwise, *WARNING* if at least one attribute is in warning
- otherwise, *CHANGING* if at least one is changing
- otherwise it is *VALID*.

To simplify the use of alarm attributes in another alarm formula the following syntax is supported:

- *name/of/device/almattr.alarm,* which evaluates true if it is equal to *UNACK* or *ACK*
- *name/of/device/almattr.normal,* which evaluates true if it is equal to *NORM* or *RTNUN*

With one attribute for each alarm, and the additional syntax described above, it is possible to build a hierarchy of AlarmHandlers as depicted in Fig. 3.

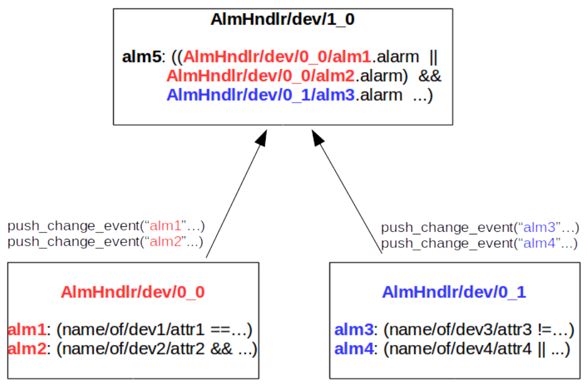

AlmHndlr/dev/1_0
alm5: ((AlmHndlr/dev/0_0/alm1.alarm ||
AlmHndlr/dev/0_0/alm2.alarm) &&
AlmHndlr/dev/0_1/alm3.alarm ...)
push_change_event("alm1"...)
push_change_event("alm2"...)
push_change_event("alm3"...)
push_change_event("alm4"...)
AlmHndlr/dev/0_0
alm1: (name/of/dev1/attr1 ==...)
alm2: (name/of/dev2/attr2 && ...)
AlmHndlr/dev/0_1
alm3: (name/of/dev3/attr3 !=...)
alm4: (name/of/dev4/attr4 || ...)

Figure 3: AlarmHandlers hierarchy.

## PERFORMANCE

Currently one instance of the novel AlarmHandler device is under test at Elettra. Almost 1000 alarms are configured, with very different number of attributes involved in the formula, spanning from just one to more that 30. Alarms have been evaluated ~10000 times in ten minutes of execution. An alarmFrequency attribute has been implemented that returns the number of times each alarm has been evaluated; the distribution for the test described above, shown in Fig. 4, has a peak of ~200 evaluations for one alarm.

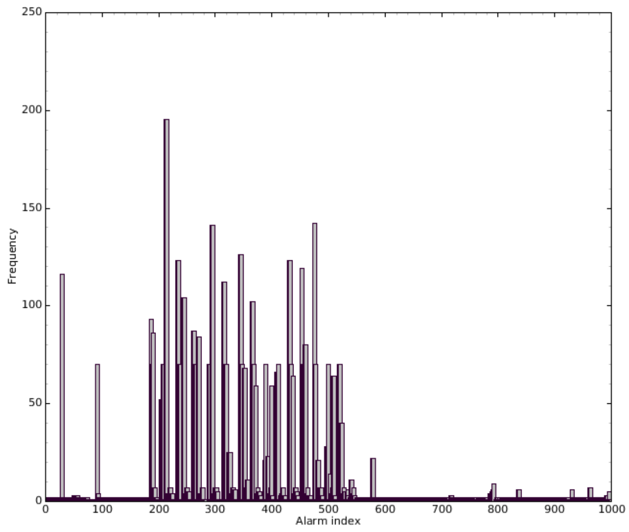


Figure 4: Number of times each alarm has been evaluated in 10 minutes.

Some attributes in the Elettra/FERMI control systems are configured to push events 50 times per second: in no case events are missed, and alarm formulae are always evaluated when triggered.

The CPU load of the AlarmHandler is comparable to the load of similar device servers running on the same machine. In details, the CPU load is not more than 1% when no polling is configured on the device, when running on a virtual machine configured to use 8 cores and 4 GB of RAM. The hardware running virtual machines is a recent industrial PC with Intel Xeon CPU, model E5-260 v2, with a 2.60 GHz clock. Around 200

TANGO device servers, mostly written in C++ and some in Python, are running on the same virtual machine, and the global CPU usage is reported to be idle for more than 90% of the time.

## PANIC INTEGRATION

A comparative analysis of the existing Elettra Alarm System and PANIC [6], the Python-based Alarm System developed at Alba, has been carried out. The minimum set of functionalities to be supported, and the common API for clients have been identified, and are reported in Table 1, together with the current implementation in the two tools.

Table 1: AlarmHandler-PyAlarm Comparison

| | **AlarmHandler** | **PyAlarm** |
|---|---|---|
| **Alarm configuration** | Attribute properties | Device/Class/free properties |
| **Alarm attribute names** | "name", "formula", "level", "message" | "tag_name", "formula", "severity", "description" |
| **Alarm grouping method** | grouped using "group" alarm property | grouped separating alarms belonging to different groups in different PyAlarm instances |
| **Alarm actions and notifications** | properties "on_command" and "off_command" used to specify commands to be executed in the NORMAL -> ACTIVE and ACTIVE-> NORMAL transitions. These commands can be used to send email notifications. | property "receivers" that can be: ACTION, an email address or a phone number. ACTION(state, expression) is used to execute the expression (sending commands or writing atributes) when the state specified is reached. Email address or a phone number are used to send notifications on every alarm state change. |
| **Interface: enum values** | NORM, UNACK, ACKED, RTNUN, SHLVD, DSUPR, OOSRV | boolean attributes |
| **Interface: attributes and commands** | one DevEnum attribute per alarm + one string array per each alarm state with the list of alarms in that state | one bool attribute per alarm + string arrays with coded information in it and list of alarms in that state |

A study, with the purpose of providing some commonalities between the two alarm system implementations, is ongoing. With the pre-requisite to be compliant to the IEC 62682:2014 standard, a first analysis lead to the following results:

- AlarmHandler and PyAlarm store the alarm configuration in different ways, but this does not affect the functionality
- alarm attributes names need to be shared; the following minimal common set has been defined, according to the standard: "tag", "formula", "priority", "message"
- both grouping method should be supported: using a 'group' alarm attribute and separating alarms in different AlarmHandler devices
- AlarmHandler needs to support a more general "annunciators" property than just "on_command"/"off_command"; a syntax like the one supported by PyAlarm should be implemented in order allow actions and notifications on every state transition
- PyAlarm needs to support the same enum values and labels defined by the IEC 62682:2014 standard for dynamic alarm attributes, as AlarmHandler do
- To support generic alarm GUIs both systems provide the *alarmSummary* attribute and the *GetAlarmInfo* command.

## CONCLUSION

Refactoring the TANGO Alarm System improved performance and functionalities, and aligned the tool to a well established international standard. Moreover, the TANGO collaboration will benefit from the standardisation effort between the AlarmHandler and PyAlarm, a work that has just started and will continue with the integration of additional functionalities. The upgrade of the Alarm graphical user interface developed at Elettra, base on QT, is also foreseen.

## REFERENCES

[1] L. Pivetta, "Development of the Tango Alarm System", ICALEPCS'05, Geneva, Switzerland (2005), WE3B.1-70

[2] The Boost.Spirit Library, `http://boost-spirit.com`

[3] L. Pivetta *et al.*, "HDB++: a new archiving system for Tango", ICALEPCS'15, Melbourne, Australia (2015), WED3O04

[4] S. Rubio-Manrique *et al.*, "Extending alarm handling in Tango", ICALEPCS'11, Grenoble, France (2011), MOMMU001

[5] "Management of alarms systems for the process industries", IEC 62682 (2014)

[6] S. Rubio-Manrique *et al.*, "PANIC and the Evolution of Tango Alarm Handlers", ICALEPCS'17, Barcelona, Spain (2017), TUBPL03