

DYNAMIC ATTRIBUTES AND OTHER FUNCTIONAL FLEXIBILITIES OF PYTANGO

S.Rubio-Manrique, T.Coutinho, R.Suñé (CELLS-ALBA Synchrotron, Barcelona, Spain),
E.Taurel (ESRF, Grenoble, France)

Abstract

Abstract ALBA, member of the Tango Collaboration, is a third generation Synchrotron under construction near Barcelona. Development of ALBA Control System soon required of highly customizable interfaces for the multiple PLCs, Vacuum and DAQ equipments being tested. On-the-run dynamic attribute creation, customized calculations, configurable state composing and attribute-grouping have been achieved applying Python; a dynamic object-oriented language with an easy syntax accessible to operators. Other new features, such as multiple device classes inheritance, have been added to the Control System and allowed integration of HW API's and high-level tools in the same process. PyTango, the Python API of Tango, is actually the common platform for most of User and Hardware interfaces developed at ALBA.

INTRODUCTION

Many features of modern languages—like dynamic typing, garbage collection and scripting—have been explicitly focused on reducing the developing time, increase readability of code and make easier the life of the programmer. In addition, many features of pure functional languages (Haskell, Lisp) have been reintroduced to mainstream developments.

Python—upon ruby, C# and other “new” languages—has achieved the highest degree of transversality; providing rich GUI's, web frameworks, database management and hardware management on any platform. This success has been achieved thanks to its easy integration with existing C++ resources (using SIP, SWIG or Boost bindings), allowing python to fill the gap between the user interface and the hardware oriented worlds.

PyTango at Alba

Advantages of Python have been intensively applied in the design, prototyping and simulation of the control system of a new synchrotron [1]. The Alba Light Source—being built in Barcelona, Spain—will start its commissioning during 2010 with most of its control applications based in PyTango; the python binding on top of Tango Control System [2].

In the same way that one of the reasons of Tango development was the necessity of a Control System that matched completely within the Object-Oriented paradigm, PyTango now offers a chance to apply new computing

paradigms and extend Tango in the field of soft and functional programming.

PyTango allowed to dynamically add new code to existing devices, loading new functions, classes and modules during startup (Multiple Inheritance); adding new variables and channels to a running device (Dynamic Attributes) or modifying its state-machine behaviour (Dynamic States and Qualities). This report describes some of the applications of such techniques in the Alba control system. Methods and classes described can be obtained from **PyTango_utils** package, available at Tango-DS repository in sourceforge.

MULTIPLE INHERITANCE

The standard approach to integrate multiple devices in Tango is the multi-class Device Server, in which several Tango Device Classes are embedded in a single executable to perform together a single class.

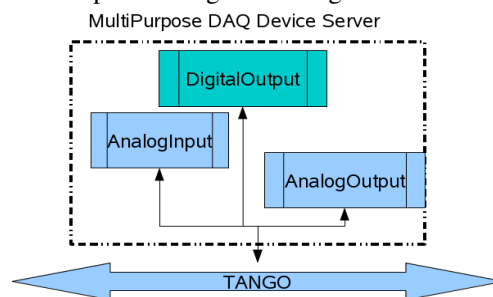


Figure 1, a multi-class Device Server[3]

In some applications this approach presented some redundancies as configuration may be duplicated in the database (each device become a separated entity) and it's needed a messaging system between the devices to coordinate their tasks—using the standard Tango Client layer or an integrated messaging framework[4].

To reduce these redundancies we added an alternative interface-based approach, assuming that classes being merged are simply adding their functionalities.

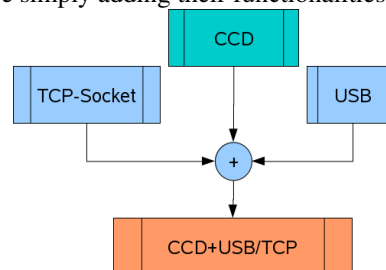


Figure 2, the Tango Interface approach, only one of the communication interfaces will be instantiated.

Class merging is done at runtime—using the built-in **type** class—, as Python allows “online” modification of class objects and its inheritance before instantiating the Tango Devices. If desired, the new class may receive the name of the original parent (CCD in the example) to replace it .

Application of multiple inheritance between Tango Classes allows, using abstract classes, to fix common behaviours and create customized versions of existing classes.

DYNAMIC ATTRIBUTE CREATION

The C++ and Java Tango Device Classes (software object used to control a hardware device) already allowed to create and remove attributes when needed [5]. In that way data acquisition/extraction devices are able to generate as many channels as needed by each specific application or hardware resource.

Dynamic attribute creation in C++ device servers is restricted to generating new copies of a template attribute with fixed read/write methods. During the process of Alba control system design and prototyping it was foreseen that further customization would be needed to adapt dynamic attributes to devices with an heterogeneous set of channels (digital i/o, analog i/o, serial ports,). This has been achieved for any device inheriting from new **DynamicDS** class.

Configuring DynamicAttributes

Declaration of dynamic attributes for PyTango Device classes can be done in several ways, either through an external command to the device or by specifying it in the Device Properties to be created at Startup.

Table 1, Tags used in Dynamic Attributes

Tag	Value
t	Seconds since device startup
READ	True for read access
WRITE	True for write access
VALUE	Value passed by client
[AttributeName]	Last value of attribute
ATTR(name)	Force attribute update
PROPERTY(name)	Value of property
EVAL(PROPERTY(name))	Evaluation of property
STATE([value])	Read or set State
STATUS([value])	Read or set Status
XATTR(device/attr)	Read other device attribute
VAR(name,[value])	Read or store a variable

The format for declaring the Dynamic Attributes is to parse a Python expression using the **eval** method of the Python interpreter. The scope of variables available for the evaluation of an attribute includes all the desired device commands and/or attributes and a set of tags that allow to access the device configuration and state. New commands are introduced modifying a **locals** dictionary within the device object.

DynamicAttributes

```
ATT=Comm1 if READ else Comm2(VALUE)
```

Table 2. Attribute declaration

Using Tango Types

Not all Tango Types are available yet, just Scalar and Spectrum Tango types can be generated. Although type can be automatically parsed it's sometimes needed to use a conversion function; convenient methods have been added for common Tango types.

Tango Types	Python
DevShort(x)	int(x)
DevLong(x)	int(x)
DevFloat(x)	float(x)
DevDouble(x)	float(x)
DevString(x)	str(x)
DevVarShortArray(x)	map(int,x)
DevVarLongArray(x)	map(int,x)
DevVarFloatArray(x)	map(float,x)
DevVarDoubleArray(x)	map(float,x)
DevVarStringArray(x)	map(str,x)

Table 3, List of Tango Types and conversion functions.

DYNAMIC QUALITIES AND STATES

Once the generation of readable values became customizable, it was needed to integrate their values within the State Machine, modifying the State/Status of the Tango device or the quality/allowance of certain attributes.

These customizations are always loaded at start-up using *DynamicStates* and *DynamicQualities* properties. In both cases the tags available are the same than those used for DynamicAttributes.

Standard Tango Device Servers provide a **hook** method that allows to perform State update before any attribute reading or command execution. DynamicDS hook method will evaluate DynamicStates declarations and will set the State to the first State that evaluated to True.

DynamicQualities for each attribute are evaluated at reading time, using previously generated values and tags to assign a value between VALID, INVALID, WARNING, ALARM or CHANGING.

Table 4. State/Qualities declaration, using an attribute that stores its modification time when it's written. State changes if value is above 10.0; quality changes for 10 seconds after writing (t is the number of seconds after start up).

DynamicAttributes
Att_X=VAR('x') if READ or (VAR('x', VALUE) and VAR('t0',t))
DynamicStates
ALARM=(Att_X > 10.0) ON=1
DynamicQualities
Att_X=VAR('t0')+10>t and CHANGING or VALID

EXISTING APPLICATIONS

PLC Device Servers

Alba EPS is an heterogeneous collections of PLC subsystems that, although sharing the same communication protocols, requires the generation of a big number of variables that vary too much.

An automatic way of generating the attributes to be read for each PLC is needed, and PyTango is the perfect tool as it allowed to create Tango Attribute declarations using the information retrieved from our cabling and connections MySQL database [6][7].

All the commands needed for accessing Modbus variables (Read/WriteRegisters) and PLC variable type (DigitalInput/Output, Valves) were implemented and became available to generate new attributes when needed.

PySignalSimulator

This Python Device Server allowed to produce any kind of signal needed for simulation or GUI testing by adding common mathematical signals to the scope of the evaluation (ramp(t), sin(t), cos(t), exp(t), triangle(t), square(t,duty), random()).

PyStateComposer

Adding two new tags to attributes evaluation, DEVICES and STATES, this Tango device helped to summarize not only the state of the different Tango subsystems (loading a pre-defined list of devices), also some key attributes if necessary.

Table 5, Composed Pressure and State

DynamicAttributes
AllPressures=[XATTR(dev+'/Pressure' for dev in DEVICES)]
DynamicStates
ALARM=ALARM in STATES ON=1

PyAlarm

This device server extended the approach used for DynamicStates to evaluate sets of conditions for different alarms. It have been used intensively during machine installation for its versatility to change alarm conditions when needed.

CONCLUSSIONS

The high adaptability of dynamically programmed objects made easier the task of prototyping and simulate entire subsystems of the accelerator. This fact allowed to start commissioning with many of the key applications already tested and also helped to detect the flaws of the system at early stages.

Dynamic attributes and other dynamic solutions may be not adequated for stable systems, but it has been clear that during the design and pre-commissioning of an accelerator—while many things are continuously changing—became a valuable tool.

In addition, the simplification of single and multi-class inheritance in python device servers allowed to apply Abstract Device Patterns [8] to PyTango device servers, a fact that should increase class reusing in the future.

Collaboration

This work would have not been possible if not highly influentiated by the work, ideas and suggestions of Roberto Ranz, Alejandro Homs, Sergi Blanch, Guifré Cuní, Carlos Pascual, Fulvio Becheri, Lothar Krause, Nicolas Leclercq and David Fernández.

REFERENCES

- [1] D.Fernández et al. "Alba, a Tango based Control System in Python", ICALEPCS 2009, Kobe, Japan.
- [2] 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.Ounsi, "TANGO a CORBA based Control System", Proceedings of ICALEPCS 2003, Gyeongju, Korea
- [3] S. Rubio "Abstract Classes for Data Acquisition", 2006, Tango Meeting, ESRF, Grenoble, France
- [4] N. Leclercq "Writing a SubSystem Device Server Manager using YAT", 2009, Tango Meeting, ESRF, Grenoble
- [5] R.Sune, E.Taurel and S.Rubio, "Adding Dynamic Attributes to a C++ Device Server", available at www.tango-controls.org
- [6] D.Beltran et al. "Alba Control & Cabling Database", ICALEPCS 2009, Kobe, Japan.
- [7] D.Fernández et al. "Alba, The PLC based protection systems", ICALEPCS 2009, Kobe, Japan.
- [8] A.Götz, "Abstract Device Pattern and Tango", ICALEPCS 2007.