PyPLC, A VERSATILE PLC-TO-PC PYTHON INTERFACE

S. Rubio-Manrique, G. Cuní, D. Fernandez-Carreiras, Z. Reszela, A. Rubio,
CELLS-ALBA Synchrotron, Barcelona, Spain

Abstract

The PyPLC Tango Device Server provides a developer-friendly dynamic interface to any Modbus-based control device. Raw data structures from PLC are obtained efficiently and converted into highly customized attributes using the python programming language. The device server allows to add or modify attributes dynamically using single-line python statements. The compact python dialect used is enhanced with Modbus commands and methods to prototype, simulate and implement complex behaviours. As a generic device, PyPLC has been versatile enough to interact with PLC systems used in ALBA Accelerators as well as to our Beamlines SCADA (Sardana). This article describes the mechanisms used to enable this versatility and how the dynamic attribute syntax allowed to speed up the transition from PLC to user interfaces.

INTRODUCTION

ALBA[1], member of the Tango Collaboration[2][3], is a third generation Synchrotron in Barcelona, Europe. It provides light since 2012 to users through its 7 beamlines,with 2 more under construction.

Programmable Logic Controllers from several vendors (B&R, Pilz, …) are used for acquisition, protection and motion within our Tango Control System[4]. PLC's are the main component of equipment and personnel protection systems, but they are also used in accelerators and beamlines for vacuum/temperature diagnostics and motion control. The most complex system managed by PLC's is the Equipment Protection System (EPS)[5].

Equipment Protection System at ALBA

The EPS is an autonomous system ensuring the safe operation of all elements in ALBA accelerators and Beamlines. It generates both interlocks and operation permits, following the logics previously defined between the Control section and the Accelerators and Experiments divisions and programmed by Control engineers.

EPS uses 58 B&R CPU's and 110 periphery cabinets to collect more than 7000 signals. In addition to the main purpose of protection, several hundreds of signals distributed across the whole system are acquired for diagnostics and control of movable elements: temperatures, vacuum sensors, position encoders and switches, electrovalves, ...

Other PLC-based Systems

The Modbus protocol and Tango devices are also used to control PLC's in the RF circulators, bakeout controllers, water cooling system, air conditioning in the experimental hutches and overall Personnel Safety Systems, on which the Tango Control System have just read access. The same control interface is used to communicate with all this subsystems, with certain customization depending on the control needs.

PLC TANGO DEVICES

An interface between Tango Control System and our PLC-based subsystems was needed for three main purposes:

- Supervision of autonomous systems based on PLC's (EPS, PSS).
- Configuration of the EPS settings and interlock thresholds during commissioning.
- Integrate critical and diagnostics signals into our Control services like Archiving, Taurus UI, Alarms, Beamlines SCADA (Sardana) [6][7].

To achieve a successful integration of the PLC signals into our control system it was needed to automate the creation of Tango Attributes in the PLC device servers. The commissioning work-flow required a regular update of I/O and variables lists in the PLC's, and existing UI's and services like archiving had to be capable to keep pace with each update of the attribute list.

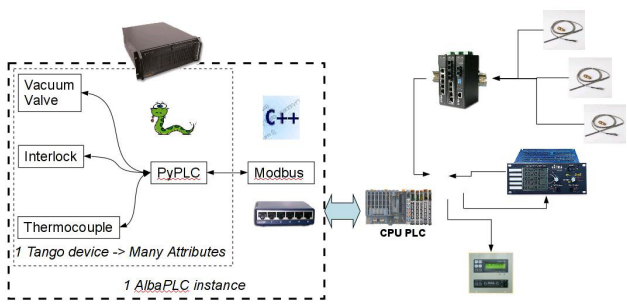

Figure 1: PyPLC Device Architecture.

DEVELOPING A PLC TANGO DEVICE

The first device server developed at ALBA for communicating with PLC's was a C++ device server, ModbusPLC, running on top of the Modbus Device developed at the ESRF; that already implements all basic Modbus commands. The ModbusPLC C++ Tango device allowed to create and remove attributes depending on Tango property values [8], exporting as many integer attributes as 16 bits registers were mapped in Modbus.

This implementation presented several drawbacks:

- Hides the diversity of signals available from the PLC (digital inputs, flag registers, integers, 16 bits floats, 32 bit values spanning multiple registers).
- Too rigid for showing complex elements (needed many attributes to represent a 3-position valve).

ISBN 978-3-95450-146-5

- Triggered 1 Modbus command per attribute read, for a typical beamline PLC with hundreds of attributes it meant a full refresh every 30 seconds or so.

PyPLC, a Dynamic PyTango Device Server

PyTango, the Python binding of Tango, is the common framework of development at ALBA [9]. The PyPLC python device server overcomes the limitations of ModbusPLC, providing the high versatility of python [10] above the robustness of the reliable Modbus C++ server (Fig.1).

Advantages of PyTango device servers in python are:

- Python is a dynamically typed language, making devices prototyping and extension much faster.
- Objects and libraries are mutable, multiple inheritance and classes are modifiable in runtime.
- Device servers can be executed in any OS with no need of compiling or packaging.
- Tools like SWIG or Boost allow to use C++ APIs libraries from python.
- In fact PyTango is just a layer above Tango C++, which means that any feature/patch of mainstream Tango is also available in PyTango.
- Python allows to execute string formulas as python code, enabling complex Dynamic Attributes declaration on runtime.

Taking profit of these advantages, we developed a DynamicDS[11] template for Tango Devices that provided dynamic attribute type creation, customized calculations, configurable state composing and attribute-grouping. All these features being achieved through an easy syntax accessible to machine operators and scientists.

PyPLC inherits from DynamicDS template [12]. Creation of attributes is done writing new attribute formulas into the DynamicAttributes property of the Tango database, a process that can be done manually or automated by scripts.

Table 1: PyPLC Attribute Formulas, Int Array and a Writeable Boolean Flag.

```
TEMPERATURES=
      DevVarLongArray(Regs(7800,100))
```

```
DIO_01= bool(
READ and Flag(80,7) or
WRITE and WriteFlag(81,7,int(VALUE)))
```

To enable the use of Modbus commands in the attribute formulas, the PyPLC exports all the commands needed for accessing Modbus variables (Read/Write Input/Holding Registers) and PLC variable type (Bit/Coil/Flag/Int/Long/Float/IeeeFloat/Double). Those methods can be used inside attributes (Table 1) or commands (Table 2) declaration to access any type of variable mapped in Modbus addresses.

Table 2: PyPLC Command Formulas

```
Open_PNV01=(WriteBit(193,2,1),1)[-1]
```

```
Close_PNV01=(WriteBit(193,1,1),0)[-1]
```

Optimizing Modbus Communications

The Modbus devices used at ALBA (mostly B&R plc's) show several limitations in the implementation of their Modbus communications. The maximum amount of registers to be acquired in a single modbus exchange is 120. And those communications take between 120 ms and 160 ms independently of the number of addresses read on each command.

To avoid these problems, PyPLC enhances the Modbus Tango Device providing smart mapping of the PLC memory. It allows to optimize ethernet/ serial communications and setup selective address refresh when needed. The memory areas are read and allocated in the device server as arrays, each of them dedicated to a certain type of data (boolean, int, float, double). Attributes of the PyPLC device will not access the Modbus communications but this arrays mapped in memory instead.

Actually the refresh of a typical ALBA PLC is about 3 seconds for a PLC with 2000 registers mapped to 300 attributes. Although faster refresh of certain areas of the memory can be setup up to 300 ms.

EXTENDING PYPLC

Exporting PLC Variables to PyPLC

When programming the PLC's, the controls engineer load variables information retrieved from our cabling and controls MySQL database [13]. Visual basic macros are used to update the PLC program using pre-defined logic blocks and the set of signals connected to each CPU and peripherals.

Those same Visual basic macros generate .csv files that are used by the EPS Taurus UI and PyPLC device server to load variables lists and Dynamic Attributes declaration.

Thanks to that, Taurus User Interfaces and PyPLC attributes are updated whenever the PLC program is modified, with no need of editing the source code.

Many analog and digital signals in ALBA beamlines have been exported to the experiment control framework, Sardana. The Sardana suite allows to add Tango controlled hardware as experimental channels, being able to use them as I/O or experimental data (e.g. temperatures and vacuum pressures). Limited motion control have been implemented using PyPLC, which allowed direct control from Sardana macros used by scientists.

ISBN 978-3-95450-146-5

Extending PyPLC Functionality

PyPLC provides a common interface to any device that uses the Modbus protocol, specific States and Status messages can be linked to PLC values using the same syntax of dynamic attributes. But, certain subsystems require a more specific interface to express intermediate states like warning, external interlocks, complex error codes, attributes managed by multiple registers (e.g. multiple-stage pneumatic elements).

This is achieved subclassing PyPLC into new Tango Devices that extend its functionality, in the same way that PyPLC inherits from DynamicDS. Those classes are AlbaPLC (customized to EPS state machines), PLCValve for pneumatic valves and FSOTR for 3-position fluorescence screens. Those classes use Tango Qualities to complement the raw attribute values, passing the information regarding Alarm/Warning/Moving limits and positions with each value sent to clients.

Higher-level procedures involving several Tango devices are implemented using macros from PANIC Alarm System[14][15] or Sardana SCADA. Those macros allow the scientists to program automated actions on elements controlled by the EPS (valves, shutters) during experiments.

CONCLUSSION

PyPLC Tango Device provides a common interface to all PLC's at ALBA using the Modbus protocol. This developer-friendly interface allowed dynamic and effort-less integration of new PLC signals into our Archiving, Alarm System and Beamlines SCADA (Sardana). This work-flow enabled by PyPLC reduced the time needed to upgrade or modify PLC systems in Beamlines.

ACKNOWLEDGEMENTS

Many former ALBA engineers have collaborated in the PLC-related projects in the last 6 years: D.Fernández, A.Rubio, R.Ranz, R.Montaño, R.Suñé, M.Niegowski, M.Broseta and J.Villanueva. The collaboration of Tango core developer, Emmanuel Taurel, was fundamental in the development of Dynamic Attributes templates and debugging of PyPLC performance.

REFERENCES

[1] ALBA website: http://www.cells.es
[2] TANGO website: http://www.tango-controls.org
[3] A.Götz, E.Taurel et al., “TANGO V8 – Another Turbo Charged Major Release”, ICALEPCS'13, San Francisco, USA (2013)
[4] R.Ranz et al., “ALBA, The PLC based Protection Systems.”, ICALEPCS'09. Kobe, Japan (2009)
[5] D.Fernández-Carreiras et al., “Personnel protection, equipment protection and fast interlock systems”, ICALEPCS'11, Grenoble, France (2011)
[6] T.Coutinho et al., "Sardana, The Software for Building SCADAS in Scientific Environments", ICALEPCS'11. Grenoble, France (2011)
[7] SARDANA website: http://www.sardana-controls.org
[8] R.Sune, E.Taurel and S.Rubio, “Adding Dynamic Attributes to a C++ Device Server”, available at www.tango-controls.org (2008)
[9] D.Fernández et al., “Alba, a Tango based Control System in Python”, ICALEPCS'09, Kobe, Japan (2009)
[10] S.Rubio et al., “Dynamic Attributes and other functional flexibilities of PyTango”, ICALEPCS'09, Kobe, Japan (2009)
[11] Fandango website: http://www.tango-controls.org/Documents/tools/fandango/fandango
[12] PyPLC website: http://www.tango-controls.org/device-servers/alba/pyplc-device-server/
[13] D. Beltran et al., “ALBA Control And Cabling Database”, ICALEPCS'09, Kobe, Japan (2009)
[14] S.Rubio et al., “Extending Alarm Handling in Tango”, ICALEPCS'11, Grenoble, France (2011)
[15] S.Rubio et al., “PANIC, a Suite for Visualization, Logging and Notification of Incidents”, PCaPAC'14, Karlsruhe, Germany (2014)

ISBN 978-3-95450-146-5