THE ALBA CONTROL SYSTEM PACKAGE SARDANA

Authors: Jörg Klora, Tiago Coutinho

Abstract: The control system architecture used and partly developed at ALBA is described. The importance of centrally stored reusable procedures and a good separation of the individual components are emphasized. Keywords are Python, C++, Qt, TANGO, and SPEC.

Introduction

This paper describes the architecture and the individual components of the ALBA control software Sardana. The names is chosen after the Catalan circle dance which according to Wikipedia (http://www.wikipedia.org) does not require special fitness but needs very good coordination between the dancers. The system consists of parts developed at ALBA, parts developed in collaboration with other synchrotrons and some parts are just very small layers on top of the work of others. The system is programmed mainly in Python and C++ and runs on Linux and in the future on Windows.

Traditionally these papers center on the hierarchical structure of the underlying devices in order to obtain higher and higher level components. We believe that it is beneficial for our control architecture to concentrate on reusable procedures. These procedures are stored and execute centrally in the system. Another important aspect was to build a system out of individual components which can all be shared separately, but also hide this complexity to the end users of the complete system. At the current stage of development this goal has only partially been reached.

ALBA is part of the TANGO collaboration. We often access the hardware via TANGO device servers. TANGO also provides us with the glue to assemble the different parts of the system. It also provides us with a rich set of tools (see http://www.tango–controls.org). We have however from the beginning taken steps to boost collaboration with partners who do not use TANGO and make components independent of TANGO.

The following image gives an overview of the complete system with its individual components.

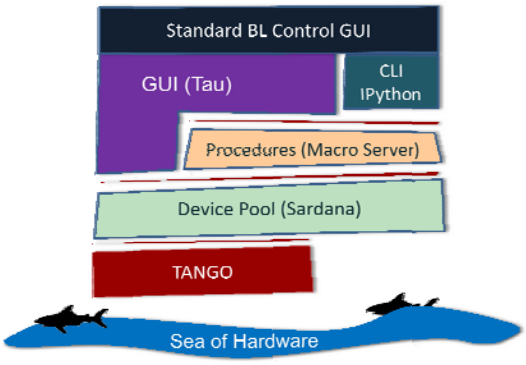

Figure 1: Overview of the ALBA control system architecture

Central Procedures

As mentioned above, procedures are stored inside the somewhat misleadingly named macro server. Every “macro” is a python

class with methods to run and possibly abort the procedure. Procedures can be added and edited with revision control from any client in the system. The arguments to the procedure are defined in the class and have a specific type. Parameter information and documentation can be accessed from the clients to construct generic input panels or for command line completion. The procedures should be as short as possible and use external independent python modules whenever possible. E.g. we have implemented our scan procedures as individual procedures, but discussions have been started with DESY, the ESRF, and Diamond to develop an independent module as a framework for continuous scans and step scans. Another example is the data flow out of the procedures. We distinguish if data is destined for on-screen watching (like standard output), plotting of the data (which is currently done by putting it into the standard ESRF shared memory format (sps<)) or the actual acquired data which flows towards further processing or to disk. The data format is decided by pluggins (currently the SPEC scan data format or the ESRF data format EDF).

Procedures can be executed sequentially or in parallel. This behavior can be controlled by the client by subscribing to different server connection called “doors”.

Procedures can only be generic when they are as independent as possible from the underlying objects. If we want to share procedures between the machine, the different beamlines and also other sites, we need a common pool of abstract items. E.g. For a scan to be generally usable it has to rely on objects with the same movable interface.

Device Abstraction and Pooling

An independent server, called Sardana Device Pool is responsible for providing abstract interfaces to hardware and implementing common tasks (like synchronization of devices or devices which only consist of mathematical formulas).

The most important feature of the Device Pool is to dynamically provide interfaces as abstract devices and play a role of an adapter to the underlying hardware. These adapters are called controllers in the language of the Sardana Device Pool. The idea is to write a controller for every different type of hardware device which will adapt this specific device to its generic device type. Let’s see this in the case of movable devices. These “movables” can be moved, scanned and so on from the procedure level but also provide events for the monitoring type of interfaces (display a position on a GUI when it changes). To add a new movable device (“motor”) the Device Pool first dynamically creates a controller from a class on disk written in C++ or Python. This controller class contains methods (e.g. start_move) with the specific code to talk to one or multiple “motors” simultaneously. These “motors” can be real motors interfaced with TANGO or with a direct connection to a serial line, or a temperature, or a quadrupole power supply.

It is impossible to write incompatible controllers. All movable devices created in this way have the same interface. It is however possible to access specific features of the controller by creating extra attributes.

In the case of a TANGO device server (e.g. a temperature controller), the original device with its natural (temperature) interface still stays fully usable. An important part of the

Sardana Device Pool is the support to easily write new controllers. Controllers can inform the Sardana Device Pool about the features their underlying hardware does not implement and the pool will try to emulate these missing features as well as possible (e.g. backlash correction is done with two different moves when not available in the hardware). The pool also tries to provide a run time configurable channel for each hardware device (examples are serial lines, gpib, or TANGO) which leaves the controller programmer the unique responsibility to implement the specific protocol of the device.

Special care has been taken to implement the control with a minimum number of hardware calls and synchronization on the lowest possible level (e.g. in hardware). The following types of devices are currently implemented: Motors (e.g. Motors, Temperatures), Counter/Timers, zero, one and two dimensional detectors (e.g. Multimeters, MCA or CCD), Registers (e.g. VME channels, attenuators, valves...), Communication Channels (Serial Line, GPIB, sockets...). Motors and Counters can also be defined by just providing the mathematical formula how they are derived. We call these devices pseudo motors and counters, but they implement exactly the same interface. We have implemented in this way a thin layer on the Soleil diffractometer code to control 2/4/6 circle diffractometers.

The Sardana Device Pool creates its components dynamically at run-time. When started for the first time, it is an empty server with only a managing device to talk to it. At any time, new controllers or new devices can be added or existing ones edited. All interested clients are informed about these changes via events and can adapt their respective functionality. Persistency of information is implemented through the TANGO data base.

Other features include “constrains” which can be put on some movements to dynamically decide if the movement is allowed. Another feature is the grouping of motors and counters. We call these counter groups “measurement groups” and make them responsible for the right synchronization of the measurement (e.g. clear the counters, start the timer...).

The Sardana Device Pool is a collaboration between ALBA, DESY and the ESRF.

TANGO

A TANGO system consists of a central database (to provide a naming service and the configuration parameters) and different device servers running mainly on Linux and Windows. It is based on CORBA. It has therefore the built-in feature that these device servers can be linked together in the same process, in different processes on the same machine, and on different machines. The central concept of TANGO is the concept of a device. A device has commands, attributes, and properties. Commands execute actions on the device and return results. They can be called synchronously or asynchronously. Attributes are the values of the device e.g. the temperature or a position of a motor. These attributes can be readable or writeable. The result of writing an attribute depends on the implementation but often means executing an action (e.g. the motor is moved when its position attribute is written). Every attribute has much important information attached like limits, warning levels, units, and so on. The properties of a device

are its configuration parameters. The TANGO system provides us with an event system to distribute information about changes to all interested clients. TANGO also provides a rich set of tools E.g. Pogo – a graphical tool to create device servers, Jive – a tool to inspect the data base, Astor – to start device servers, Archivers, or a synoptic editor to create dynamic synoptic views.

We write device servers based on abstract devices (interface definitions) for most of our commonly used hardware. TANGO also provides us with the interface as indicated with the red lines in the above diagram. TANGO is much more restrictive as a general client server communication protocol but in this way we can assure the interoperability with other systems.

CLI

The command line interface (CLI) runs stand alone or incorporated in a GUI. For our users it provides an environment with the same commands as SPEC (Certified Scientific Software http://www.certif.com). As this program is the quasi standard of control programs, the user almost immediately feels at home.

This command line interface has been implemented on top of the Sardana macro server. It uses IPython which provides all necessary features for command input, command repetition, and command and argument completion (http://ipython.scipy.org). The CLI retrieves the list of all procedures with their arguments from the macro server and keeps itself informed about all changes via TANGO events. As our basic procedures have the same name as their SPEC equivalent i.e. for moving a motor (e.g. mv th 23.3) or handling the diffractometer (e.g. ca 1 1 0), the user sees his standard interface. It might be useful at this point to stress that the procedures initiated in this way are executed on the macro server and the code for these macros is written in pure Python with an object oriented view of its components. As the code for the procedures as well as for the controllers inside the Sardana Device Pool can be edited from the command level, users of the system might never realize the complex architecture of the system.

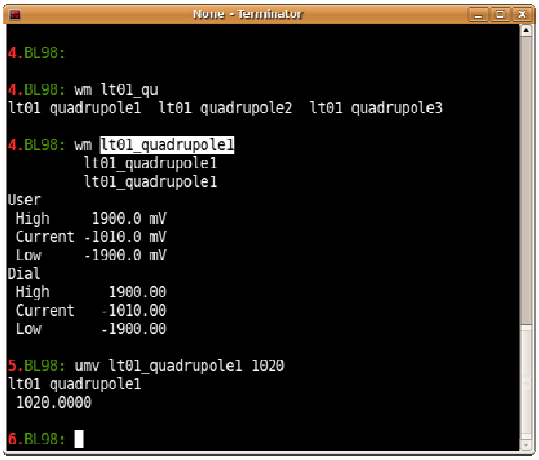

Figure 2: Example of a CLI to do a scan on a quadrupole magnet with a SPEC like interface

Tau

Tau is the graphical interface layer at ALBA. It uses Qt (Trolltech) and PyQt. It, as many of the other GUI toolkits, implements a model view controller pattern. The models are provides by the factories inside the Tau Core, which is independent of the GUI toolkit used. We have many different factories. Factories which can provide objects connected to the values or configuration parameters of TANGO attributes, objects connected to our shared memory standard (sps), objects connected to the data base, or in the future also

objects connected to Tau Core Pseudo Devices.

The basis of the Tau widgets are standard Qt4 graphical widgets written in Python or in C++ and interfaced with sip. These widgets are kept separate from the rest of the system to be able to collaborate and use them on their own. The most complex widgets are a set of graphical widgets for one and two dimensional data from the ESRF and are called Qub. We also have our own set of control widgets, widgets from the Elettra machine control system, widgets to display complex synoptic views created with the Tango synoptic editor JDraw and of course the already very rich set of widgets provided by Qt.

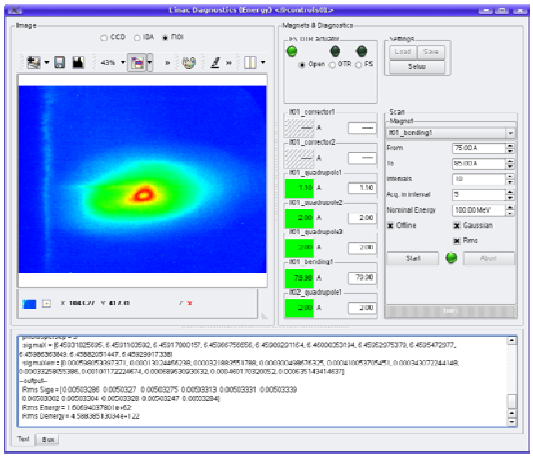

Figure 3: Example of a Tau GUI with same functionality as the CLI in Figure 2

Tau widgets are thin layers above these basic GUI widgets adding a model property, code to get the model from the factory and subscribe to changes in the values. Our models are indentified in an URL syntax (e.g. tango://id/bl/1/position?config=unit would refer to the unit of the position attribute of the device id/bl/1). We also tried not to overload the widgets with functionality but provide an extension mechanism for widgets we call actions. Application programmers can develop new actions or use standard actions to add this special functionality later.

Tau Widgets can be put together to interface screens with the powerful Qt Designer or with programming skills. Screens created in this way are immediately executable. It will also be possible to integrate them in the generic beamline application framework written at the ESRF.

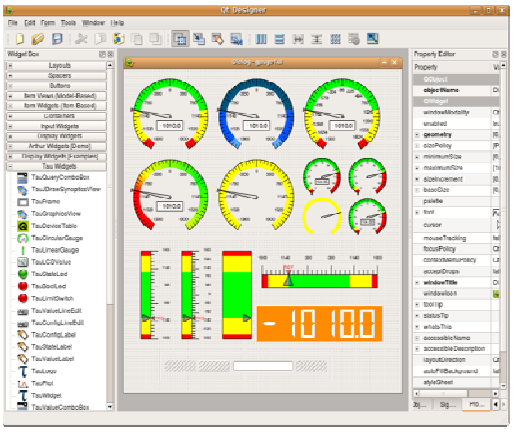

Figure 4: Example of the different versions of the Elettra gauge widget to be added with the Qt designer

There are many successful toolkits (e.g. EDM, ATK, JDDD...) often used in the control of the accelerators which produce interfaces without too much programming. They might map the hierarchy of the physical devices and flatten them (e.g. in a list of “polled” attributes) to make them accessible to the GUI. As we would like to arrive at standard interface components either programmed by hand or produced with the Qt designer we were worried about the run time configuration and a reusability of graphical panels created in this way. We are therefore currently adding a component model hierarchy where children widgets pass their model string first to their parents and then up the hierarchy. In this way, we can implement that the top widget

gets its parameters from a file or a database at runtime. It is also easier for the GUI designer to enter the different URL names as he has only to specify the missing information. An example of these components could be to assemble the GUI elements for a motor into a frame with the device name of the motor as a model name. One could then assemble two of these motor widgets together in a new component. This component would assign the underlying motor device names according to their role.

Another future addition is the possibility to write pseudo devices very easily inside the GUI. These pseudo devices will provide the attributes and commands indistinguishable to the Tau Widgets. This would facilitate the adaptation of our graphical user interfaces in situations, where one does not want to use TANGO, nor the rest of our system (e.g. at other installations).

The Beamline GUI Framework

We would like to use the new version 4 of the GUI framework in which all beamline GUIs are written at the ESRF (e.g. the well known GUI for the protein crystallography beamlines MxCube). The generic beamline GUI contains a lot of functionality which is useful on all beamlines (e.g. log-in of the users) and also a very large number of very high level GUI components. In order to reach this goal we are collaborating with the ESRF also in this area.

Our software writes data in the same format to disk and to shared memory as at the ESRF. Therefore many standalone programs developed over the past decade can be immediately used at ALBA. It is for example possible to look at and fit MCA spectra online with PyMca developed at the ESRF.

Acknowledgements

There are many authors of the software system described here. It would be impossible to name them all but let us not forget: Emanuel Taurel (ALBA now ESRF), Alejandro Homs (ESRF), Vicente Rey (ESRF), Gerry Swislow (CSS), Teresa Nuñez (DESY), Thorsten Krach (DESY), David Fernandez (ALBA), Matias Guijarro (ESRF), Gilles Berruyer (ESRF) for Qub, Sebastien Petitdemange (ESRF) for Qub, Armando Sole (ESRF) for PyMca, Frédéric-Emmanuel Picca (Soleil) for the hkl library, Claudio Scafuri and other friends at Elettra for their widgets, Richard Woolliscroft (Diamond) and Geoffrey Mant (Daresbury Laboratory) for discussions, the control group at ALBA (especially Sergi Blanch, Guifre Cuni, Sergi Rubio, and Jairo Moldes) and everybody in the TANGO Community (especially the sales director and progress pusher Andy Götz). I would also like to stress that only a very little part of the control software is the actual code for the architecture and the biggest part of the software at ALBA has been provided by a much larger group of people.