CONTROL ROOM GRAPHICAL APPLICATIONS FOR THE ELETTRA NEW INJECTOR

G. Strangolino, C. Scafuri, G. Scalamera, L. Zambon, Sincrotrone Trieste S.C.p.A., Trieste, Italy
V. Forchì, ESO, Garching bei München

Abstract

Integrating the Tango Control System with the Qt framework lead to an efficient multithreaded architecture, named *QTango*, whose components have allowed the design and implementation of Graphical User Interfaces aimed at controlling the Elettra's new injector. This paper describes the structure of the library together with some generic and specific tools taking advantage of the *QTango* infrastructure.

INTRODUCTION

Elettra is a 2.5GeV 3rd generation light source in operation since October 1993. For the commissioning and operation of the new Elettra injector [1, 2] and for the design of the future Graphical User Interfaces aimed at interacting with the FERMI@Elettra single-pass FEL user facility control system, a QT4 [3] and Tango [4] based framework has been developed. All the control room graphical applications are built upon this infrastructure, called *QTango*. *QTango* allows simple functionalities for building control panels that are Tango aware: creation of device proxies, event subscription with polling fallback in case of registration failure, device centric threads, error logs, et cetera. The communication layer sits below a QT based group of widgets wearing a pleasant, easy to use and human computer interaction oriented graphical interface.

QTANGO ARCHITECTURE

Overview

QTango is a framework founded on a set of widgets based on the *QT* libraries, named *qtcontrols*, and a Tango specific communication layer, named *qtangocore*. *QTango* combines *qtcontrols* and *qtangocore* to provide, at a higher level, easy creation of widgets ready to connect and interact with the Tango distributed control system. The full integration with the *QT4 designer*, allows a simple drag and drop of a handful of widgets onto a control panel.

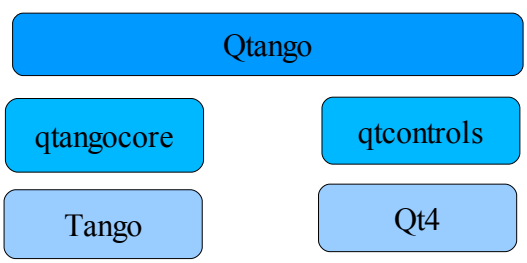

Figure 1: Representation of *QTango* underlying layers.

QTANGOCORE

Introduction

qtangocore represents the lower layer of the *QTango* framework. It is cognizant of all Tango aspects, but it is not aware of the representation of the data it is in charge of providing to its end user (e.g. the graphical objects). It is based on *QtCore threads* and *signal/slots* infrastructure. The other mainstay of *qtangocore* is represented by the libraries provided by *Tango*, allowing to create *device proxies*, to read and write on them, to destroy them when no more in use, and to subscribe for asynchronous events delivered by the *Tango* distributed framework itself.

Communication Handles Creation and TAction Objects As a Mean of Information Exchange

The interest of an object to read or write a quantity on a device of the control system is declared by the so called subscription to a *Tango source*. A *source* is identified by the *device name* plus the *attribute* or the *command* the subscriber is interested in. For example, the source *p/power_supply/psch_b11.1/Current* will allow the subscriber to read the current from the power supply identified by the sequence *domain/family/member*. Inside *qtangocore* the *QtangoComProxyReader* and the *QtangoComProxyWriter* represent the handles which an object needs to read from or write to a *tango device*, respectively. The proxies mentioned above are designed with the purpose of providing a simple interface for a *QT* widget representing disparate quantities. They really cannot inherit from *QT*'s *QObject* (*being* a *QObject* would indeed provide the necessary *signal/slot* architecture needed to refresh the graphical interfaces), but must *have* a *QObject* handle (see Figure 5).

The *QtangoCommunicationHandle* represents the mean through which a widget or some other object who wants to benefit from *qtangocore* can set its *source* to read and write *tango attributes* or to impart *commands*. The *source* configuration process develops through a number of stages:

- a syntactical validation of the string representing the source point;
- the creation of a *subscriber proxy*, which tries accomplishing the configuration of the handle;
- the *subscriber proxy* creates *one device thread per device proxy*, allocating a new object, if none is yet existing for that source, or retrieving an already existing one;
- the *device thread* obtained is immediately asked to create, through an *ActionFactory*, a new action (a *TAction*), which is the *communicator* that *links the handle to the end user* (e.g. a *qtango* widget).

Observations

Discussing this new version of *QTango/qtangocore* one must emphasize that just one *thread* per *tango device* is created, and that this single proxy reads and writes its attributes or passes on its own commands. Moreover, if more than one reader is configured with the same *source*, reading *period* and *mode* (*polled* or *event driven*) the same *TAction* is the unique bridge from the reader (writer) and the *tango* point. Finally, the *TAction* is created and *performs its tasks inside a device thread*. The latter lives in a different thread with respect to the main one. This expedient leaves *in the background* the *qtangocore/tango* data transfer, allowing the main thread to be fully responsive for the human interaction, also in case of network or device hangup.

QTCONTROLS

qtcontrols is a library made up of a set of widgets designed to represent efficiently and user friendly the *tango* quantities to read from and write to the control system devices. Thus, one can find *labels*, *apply buttons*, *circular* and *linear gauges*, *spin boxes* and other primitive graphical objects. Some of them are simply extensions of existing *QT* widgets, whilst others are tailored to fulfil peculiar requirements, e.g. the gauges.

QTANGO

Introduction

QTango is the glue that combines *qtcontrols* and *qtangocore* with the purpose to provide at a higher level a widget suited to clearly *represent a tango value*, be it a *scalar* or a *spectrum*, with or without a measurement unit, having or not warning and alarm thresholds and, at the same time, an object that *easily configures itself* to be able to perform read and write operations on the *tango devices*. From the developer point of view, the integration of the *qtango* widgets into the *QT4 designer* grants a fast and immediate design of a control panel that manages the visualization and the dispatch of *tango* quantities, handling contextually the device, network and end user errors.

Implementation

A *qtango* element *is a qtcontrols* widget *and* a *qtangocore communication proxy reader* or *writer*. Simply inheriting from both a particular *qtcontrols* widget and *QtangoComProxyReader* or *Writer*, each *qtango* class is potentially a *tango* reader or writer. Calling *setSource()* on the composite object, with the tango *source point* as parameter, initiates, configures and starts the reader (or the writer).

In addition, a *QTango* widget can conserve the *history* of a tango attribute and display it through a plot of the tango values read over time.

Eventually, a so called *helper application* can be associated to a *QTango* object. It can be launched directly from the widget simply right clicking with the mouse.

The figure 5 illustrates all these features providing a detailed class diagram for *QTango*.

CONTROL ROOM GRAPHICAL APPLICATIONS FOR THE ELETTRA NEW INJECTOR

All the operator control panels for the *Booster* have been developed with the *QTango, Qt* and *tango* libraries. In the following subsections we provide some examples of graphical user interfaces currently in use.

GenericTool

The *generictool* is a graphical user interface that gathers and displays a subset of *tango* quantities (*attributes* or *commands*) peculiar to a *class of devices*. For instance, a class of power supplies will typically represent the *state* of the device, *display* and *set* the *current*, and provide some standard *commands*, such as *On*, *Standby*, *Off*, *Reset*. The screenshot in figure 2 shows the *generictool* that, given a *family* of power supplies (*b/power_supply*), connects to all of its *members* and dynamically builds the control interface.

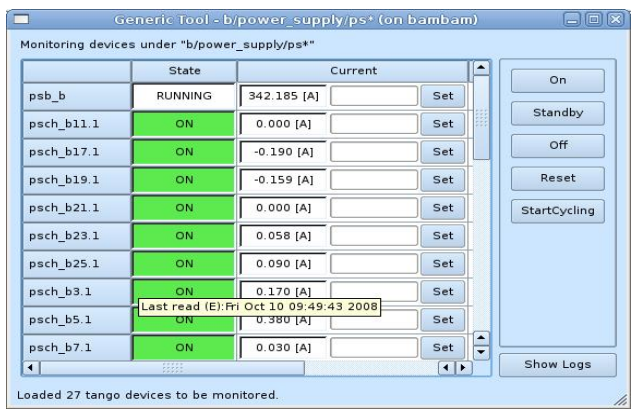

Figure 2: The *generictool* connects to a family of devices that provide a common interface.

SaveRestore

The *saverestore* control room application is in charge of making a *snapshot* at a certain instant of a *context* defined by a set of *tango* quantities. The *snapshot* of the *tango attributes* with their values is stored into a *mysql* database. Afterwards, the values saved can be *restored* on the equipments. Both at the moment of the *save* process and the *restore* one, *saverestore* warns the user if some devices are not responding or one of the processes is unable to complete correctly. The application can perform also the following operations:

- a comparison between the current values on the field and a stored *snapshot*;
- a comparison between two *snapshots*;
- a display of the *save/restore history*;
- reads and saves to text files of the *snapshots*.

A screenshot of the *saverestore* panel is available in Figure 3. The *saverestore* panel is used daily by the operators to setup all the *booster* plants for the *storage ring* refill procedure.

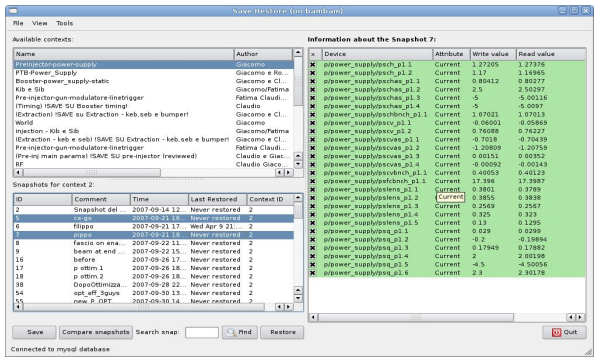

Figure 3: *saverestore control panel*.

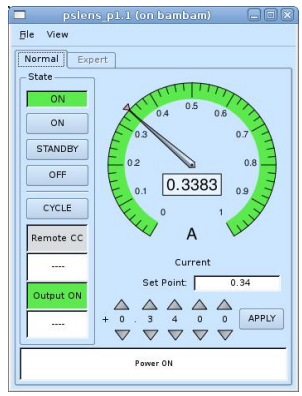

Figure 4: A sample *QTango* based application.

A simple control room panel

The figure 4 finally reports a sample control room panel, with a *gauge*, some *labels* and a widget to set the current. This is a specimen of what one is able to do by means of the *QTango* library. The panel has been designed simply dragging and dropping widgets in the *Trolltech*'s *Qt4 designer* without any additional hand written code.

REFERENCES

[1] M. Svandrlik et al., "Overview of the Status of the Elettra Booster Project", these proceedings
[2] M. Lonza et. Al, "Implementation and Operation of the Elettra Booster Control System", EPAC 2008, Genoa, Italy
[3] Qt for Application Development, http://trolltech.com
[4] Tango, http://www.tango-controls.org

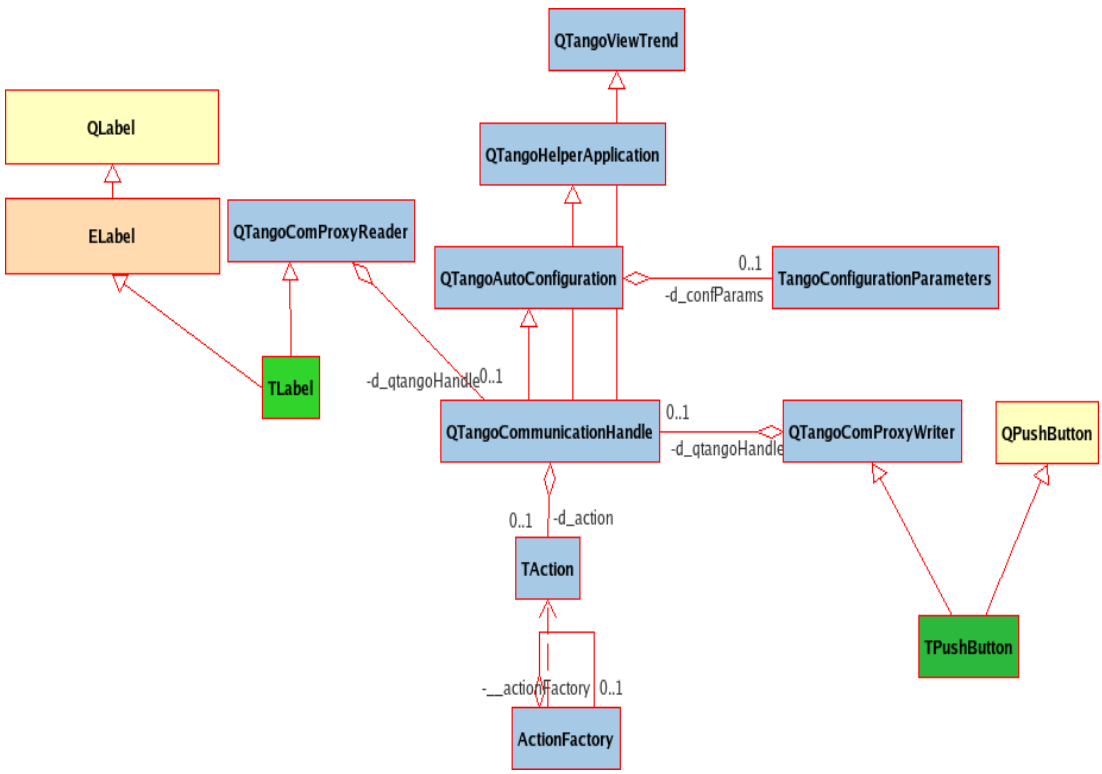

Figure 5: The class diagram of the *QTango* framework. The *QTango* widgets are represented in green color, the *qtangocore* classes are blue, the *qtcontrols* widgets are orange and the base *QT4* widgets are yellow.