QTANGO: A LIBRARY FOR EASY TANGO BASED GUIS DEVELOPMENT

G. Strangolino, F. Asnicar, V. Forchì[#], C. Scafuri, Sincrotrone Trieste, Trieste, Italy

Abstract

QTango is a framework integrating the Tango control system with Trolltech *Qt4* core and GUI libraries. It implements an efficient multithreaded and object oriented architecture, letting software developers easily and quickly compose Tango aware control system panels. Creation of device proxies, event subscription, device polling, device centric threads, error logs and other features are transparently available through the core of the *QTango* library. The latter is distributed together with a set of widgets wearing a pleasant, easy to use and HCI (Human Computer Interaction) oriented interface. The kit of *QTango* widgets includes labels, buttons, linear and circular gauges, numeric writers, spin boxes, line edits and different kind of plots. The core library allows straightforward development and integration of new graphical elements. Version 3, the new major release of the framework, is the basis for the design of the graphical user interfaces of the FERMI@Elettra Free Electron Laser control system.

INTRODUCTION

Elettra is a 2.4GeV 3rd generation light source in operation since October 1993. For the development of the graphical interfaces of the new booster injector [1] and the FERMI@Elettra free electron laser [2] a framework called *QTango* has been developed based on *Qt4* [3] and Tango [4]. Control panels built with *QTango* 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 sets below a *Qt* based group of widgets wearing a nice, easy to use and HCI oriented graphical interface.

QTANGO ARCHITECTURE

Overview

QTango is a framework founded on a set of widgets designed using the *Qt* libraries, named *qtcontrols*, and a Tango specific communication layer, named *qtangocore*. *QTango* combines together *qtcontrols* and *qtangocore* to provide, at a higher level, easy creation of widgets ready to connect and interact with the Tango distributed control system. Figure 1 represents the described architecture. The full integration with the *Qt4 designer*, allows a simple drag and drop of a handful of widgets into a control panel, immediately able to read and write quantities of any type and format from and to the Tango substratum.

[#]ESO, Garching bei München

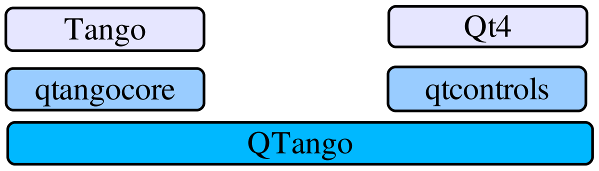

Figure 1: Representation of *QTango* underlying layers.

QTANGOCORE

Introduction

QtangoCore represents the first layer of the *QTango* framework. It is aware of all Tango aspects, but it does not know anything about the representation of the data it is asked to provide to the end user (e.g. the graphical objects). It is based on *QtCore* threads, signal/slots infrastructure and *Qt* event system.

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 and the attribute or the command the subscriber is interested in. For instance, 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 thought with the purpose of providing a simple interface for a *Qt* widget representing disparate quantities. For this reason, they cannot inherit from *Qt QObject* (which would indeed provide the necessary signal/slot architecture needed to refresh the graphical interfaces), but must contain inside them a *communication handle*, which undoubtedly is a *QObject*, implementing a *"has a"* design pattern relationship (Fig. 2).

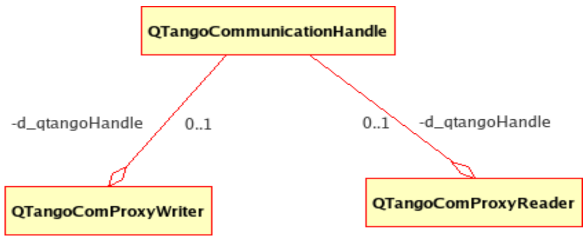

Figure 2: The *has a* relation between the *ProxyReader* and *Writer* and the *CommunicationHandle*.

The *QTangoCommunicationHandle* is privately created and owned by the *ComProxies*. It represents the mean through which a widget or some other object which wants to benefit from *qtangocore* can set its source to read and write Tango attributes or to give commands. The source configuration process develops through a number of stages:

- a syntactical validation on the string representing the source point through regular expression based filters;
- the creation of a "subscriber proxy", which tries accomplishing the configuration of the handle (in practice, creating a specific object being the real mean of communication that provides the actual data to the client, the "*Taction*");
- the subscriber proxy creates one device thread per device proxy, allocating a new one if none is yet existing for that source, or retrieving an already existing thread if previously another source was configured with the same device proxy;
- the device thread obtained (new or existing), 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 efficiently and user friendly represent 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 other 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 and handles contextually device, network and end user errors.

Implementation

A qtango element *is a qtcontrols* widget *and/or* 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 Fig. 3 one can examine the class diagram representing the *TLabel* implementation, which is a typical *QTango* widget that displays a single value, interpreting its type, measurement unit, quality of the read value (valid/invalid) and warning/alarm thresholds.

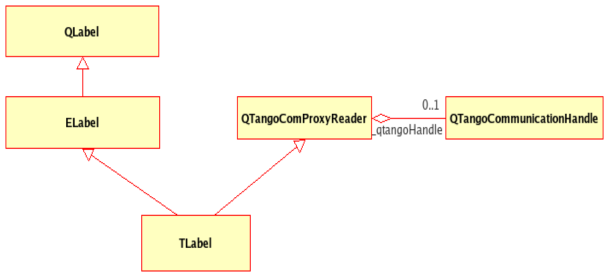

Figure 3: Class diagram for TLabel, a *QTango* widget that displays a value on a label.

As mentioned above, a widget can automatically configure itself to obtain from the Tango database the warning and alarm values of the attribute it is linked to. If such values are available the widget can configure itself to clearly display anomalous situations coinciding with critical values, normally assuming one of the three special colours green, yellow and red.

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

Eventually, to a *QTango* object a so called "helper application" can be associated. It can be launched directly from the widget by simply right clicking with the mouse. Details of the underlying communication with the Tango devices can be obtained through a "*what's this*" implementation which is also able to provide help and descriptive messages about the Tango entity represented. Drag and drop of Tango sources among widgets allows "live" creation or activation of *QTango* objects, thus realizing a sort of intercommunication among applications.

The last features described are a direct consequence of the inheritance from *qtangocore* communication proxies. In Fig. 4 a more detailed class diagram for *QTango* is provided, in case the reader is interested in understanding the full framework or in analyzing thoroughly the library implementation.

The class diagram depicted provides an example of a couple of *QTango* widgets: a *TLabel*, a reader, and a TPushButton, a writer. The key objects destined to manipulate the Tango layer are represented by the *qtangocore* *QTangoCommunicationHandle* and *QTangoComProxyReader/Writer*. As one can see, a *QTango* widget can be derived from a qtcontrols one (*TLabel* comes from *ELabel*), when peculiar data representation is needed, or from a simple *Qt* push button, as in the case of the *TPushButton*. The two classes *SubscriberProxy* and *UnsubscriberProxy* are responsible of the communication setup, and then they are immediately destroyed. For this reason, the are not owned by any other component. Observing the class diagram in Fig. 4, one can appreciate also the presence of an *Action factory*: it contains and manages the set of *TActions* (the communicators which read data and send the results to the *CommunicationHandle*) needed by the *QTango* objects. Actually one must keep in mind that two *QTango* objects referring to the same source point share the same *TAction*. Finally, in the upper part of Fig. 4, the view trend, the helper application and the auto configuration managers are represented and inherited by the communication handle. The communication handle is the element that must take care of these three aspects, being the one that holds the links to the Tango layer.

CONCLUSIONS AND PERSPECTIVES

QTango is currently used at Elettra and makes up the basis of all the control room graphical applications. The libraries are stable, efficient and easy to interact with, both for the programmer and for the end user. As for the future, the development is aimed at increasing usability and interoperability among objects and applications. A final goal is represented by the realization of an "online" designer, where widgets may be dragged and dropped from other control room panels or an internal factory and immediately connected to the Tango devices.

REFERENCES

[1] M. Lonza et al., "Implementation and Operation of the Elettra Booster Control System", ICALEPCS 07.
[2] M. Lonza et al., "The Control System of the FERMI@Elettra Free Electron Laser", these proceedings.
[3] Qt for Application Development, http://trolltech.com.
[4] Tango, http://www.tango-controls.org/.

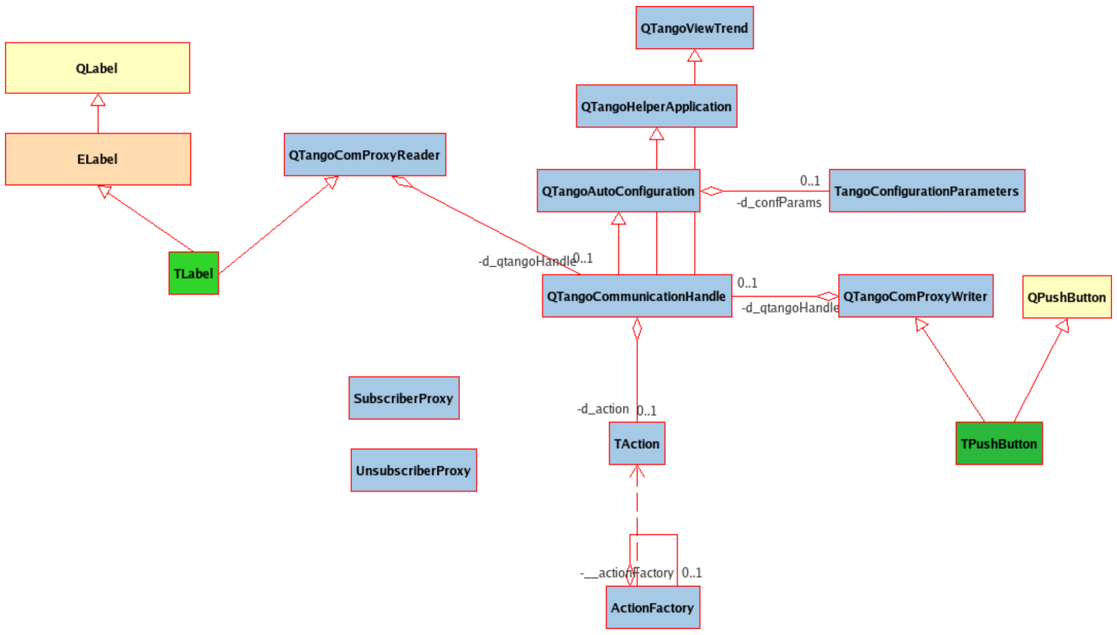

Figure 4: A class diagram representing the inheritance of two *QTango* objects: *TPushButton* and *TLabel* (green). The *qtangocore* objects are blue coloured, the qtcontrols ones are orange, while the *Qt* base widgets are represented in yellow.