# Taurus Integration to ELT Control Software

Arturo Hoffstadt Urrutia[a], Gianluca Chiozzi[a], Mario Kiekebusch[a], Carlos Pascual-Izarra[b], Zbigniew Reszela[c]
[a]European Southern Observatory (Germany); [b]iPronics Programmable Photonics, S.L. (Spain); [c]ALBA Synchrotron (Spain)

## ABSTRACT

Taurus [1] is an opensource GUI framework that implements a Model View Controller (MVC) design pattern tailored for Control Systems. It is based on python and Qt and is extensively used in the Particle Accelerators and Large Experimental Physics community and with Tango Controls.

Taurus has an active community and solves design patterns and requirements that also the ELT Control Software project shall address for its GUIs. It provides an homogeneous way to interact with any control system (Attributes and Devices) and has extension points for other projects, widgets, and factories.

A Taurus "Model" plugin adds support for a new control system, data access. It is the Model component of the MVC pattern. In the ELT case, an "oldb" model plugin was developed. This maps data-points and the tree structure of the database to Attributes and Devices, respectively. Support for read, subscription, write and polling operations were added incrementally over the versions of the "oldb" plugin. Conversely the MAL plugin supports access to the request-reply interfaces of the ELT software applications.

Once a plugin has read support, developers have access to the features the Taurus framework offers: Taurus widgets will automatically work with the new model; and scalars, vectors and matrices widgets have immediate support. New widgets for particular ELT requirements are developed as normal Qt Widgets and a Controller class is added that completes the MVC pattern.

A review of the integration is presented. An analysis of lessons learned offers our perspective of adoption. Finally, the future work in terms of GUI development is discussed.

**Keywords:** GUI, MVC, Taurus, Qt, UI, HMI

## 1. INTRODUCTION

ELT is a large telescope with a primary mirror of 39.3 meters divided into 798 segments. It is currently under construction by ESO and is located at Cerro Armazones, Atacama Desert, Chile. Its optical path is composed of 5 mirrors, it has adaptative optics, 6 laser guide stars, and multiple instruments under development. An overview of the Telescope construction is available in [2].

Telescope Control System, Instruments and Real Time Software all have the requirements to develop diverse UIs that will provide monitoring and control on various aspects of the ELT. While the functions and requirements of the different software involved may differ, the ELT has a common set of middleware and communication patterns that allows them to interact with each other. ELT Control Software has a standard set of packages denominated ECOS, which among other libraries also provides Control UI Toolkit (CUT). It relies heavily on Taurus, and provides new plugins, widgets, and UI design patterns to create new GUIs. (For a more complete description of Taurus please refer to [3])

In terms of taurus, CUT provides two scheme plugins and multiple taurus widgets. It extends certain functionalities of taurus as well. These two scheme plugins allow connection of taurus widgets to the Online Database (OLDB) and to servers using the Middleware Abstraction Layer Request Reply (MAL RR). The OLDB is based on Redis and it is used for visibility and introspection on the control system; MAL RR is used for RPC and has an implementation based on ZeroMQ with ProtocolBuffers for serialization. An update on ELT Control System is available [4].

ELT Control GUI has determined as part of its requirements that applications are to use declarative definitions of UI elements, declarative databinding, use of MVC or a similar pattern to encourage reuse, a gallery of reusable widgets

Software and Cyberinfrastructure for Astronomy VIII, edited by Jorge Ibsen, Gianluca Chiozzi, Proc. of SPIE Vol. 13101, 131013R · © 2024 SPIE
0277-786X · doi: 10.1117/12.3019717

between ELT projects, polling, subscription, compatibility with zero-code and low-code approaches, and support for multiple data sources.

Taurus is perfectly suitable for the ELT Control System as it is intended for control systems and has multiple extension points, the most attractive one and the matter of discussion in this paper: the scheme plugin.

## 2. SCHEME PLUGIN AND TAURUS MODEL

The Taurus Core module defines the Model of the MVC design pattern, or at least its interface. An important distinction for people who worked with Qt before is that Taurus MVC is not based on *QAbstractItemModel*. It is oriented for single datapoints and offers an abstraction compatible with control systems.

The Scheme plugin is an extension point for taurus. It defines how new scheme plugins are loaded and the requirements for taurus to interact with them. *tango_archiving*, *h5file*, *oldb*, and *malrr* scheme plugins make use of the defined interaction methods to be installed separately of taurus (i.e.: a separate python module) but be considered as an available supported scheme. See [5] for details on the extension point.

These plugins receive their name from the fact that they provide support for new URI schemes: Taurus Model relies on URIs to identify distinct parts of the Model. The Models provides abstractions for Authority, Device and Attributes.

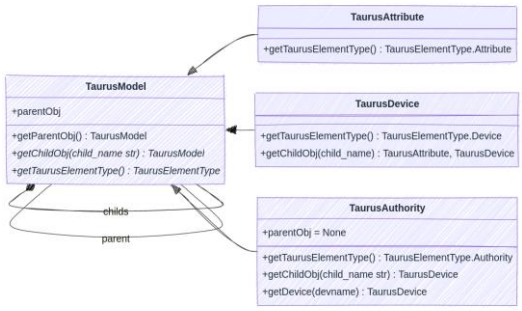


Figure 1. Taurus Model main classes

Authority acts as the root of a tree, which can have Devices and Attributes as children. Devices can also contain Attributes. Attributes act as the leaves of the tree structure.

## 3. IMPLEMENTATION

Implementation of NameValidators is the most important and requires correct and precise parsing of URIs and maps concepts of the underlying control system to Taurus Attributes, Devices and Authorities. The first step is to prepare a list of URIs that are invalid to reject them, and another of valid URIs and map them to Taurus Model concepts.

Table 1. Examples of URIs, Model Types and represented data sources/control object for cii.oldb and mal.rr scheme plugin.

| # | Model name (URI) | Model type | Scheme | Represented data source/control object |
|---|---|---|---|---|
| 1 | cii.oldb:///cut/demoservice/instance1/boolean-scalar-twosecs | Attribute | cii.oldb | Test boolean datapoint, changes state every two seconds. |
| 2 | cii.oldb:///cut/demoservice/instance1/int-scalar-add | Attribute | cii.oldb | Test integer scalar, number gets larger. |
| 3 | cii.oldb:///cut/demoservice/instance1/double-scalar-sin | Attribute | cii.oldb | Test double scalar with value of sin(t) |

| | | | | |
|---|---|---|---|---|
| 4 | cii.oldb:///elt/hlcc/telif/mon/state | Attribute | cii.oldb | State variable of HLCC Telescope Interface application. |
| 5 | cii.oldb:///elt/hlcc/telif/mon | Device | cii.oldb | Collection of all datapoints for cii.oldb///elt/hlcc/telif/** |
| 5 | mal.rr://localhost:12802 | Device | mal.rr | Demo server application, autodiscovers interfaces. |
| 6 | mal.rr://localhost:12802/StdCmds | Device | mal.rr | Demo server, assumes interface is StdCmds. |
| 7 | mal.rr://localhost:12802/Commands?if=::demoserviceif::Commands | Device | mal.rr | Demo server, forces interface used to Commands |
| 8 | mal.rr://elthlccd82:12802/StdCmds | Device | mal.rr | ELT server application in elthlccd82 host. |

Taurus recommends using the *h5file* scheme plugin [7] as the base for your new plugin. It has a simple implementation and provides an *Attribute*-centric implementation. It is not mandatory to implement all classes. It is only required to implement Factory class [6].

This was followed by the implementation of URI parsers for Authority, Device and Attribute. This is done in the Name Validators and is the most important task. Taurus provides helper testing methods that makes the implementation of test cases easy. It is important to capture from the URI regular expression any helper information that make the implementation of data identification or server host identification easier. If extra query parameters are needed, they should be captured here as well.

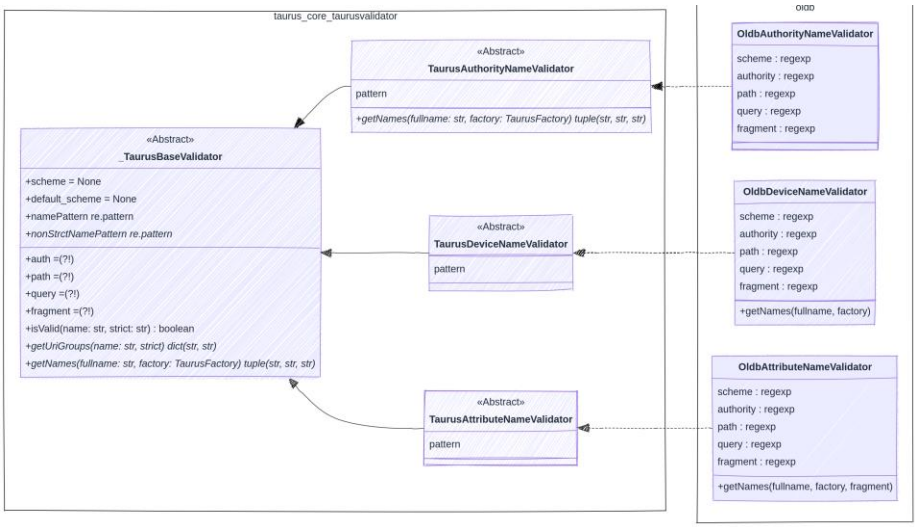


Figure 2. Name Validators Class Diagram. It includes their interfaces and abstract methods in *italic*.

Factory only creates objects that are Authority, Device or Attribute in type. It uses the Validators defined previously to check if a URI is valid, and then divide the URI into its components. Factories are expected to be able to create any of the three. The realization of Factory is listed as:

- *elementTypesMap*: The Factory implementation uses this dictionary to instantiate the correct class to an Authority, Device or Attribute of the cii.oldb scheme plugin. It maps *TaurusElementType* to a *cii.oldb* python module classes.
- *getAuthorityNameValidator*, *getDeviceNameValidator*, *getAttributeNameValidator*: All of these return an instance of the appropriate name validator for this scheme plugin.

A special case in the *cii.oldb* scheme plugin is the implementation of *findObjectClass()*. The URIs offer no distinction in the notation for Attributes or Devices. A URI could be either. *findObjectClass()* makes a programmatic distinction between both by querying the OLDB server if it has children datapoints. Though it would be ideal to make the distinction by URI, it would break the existing specification of the OLDB URIs.

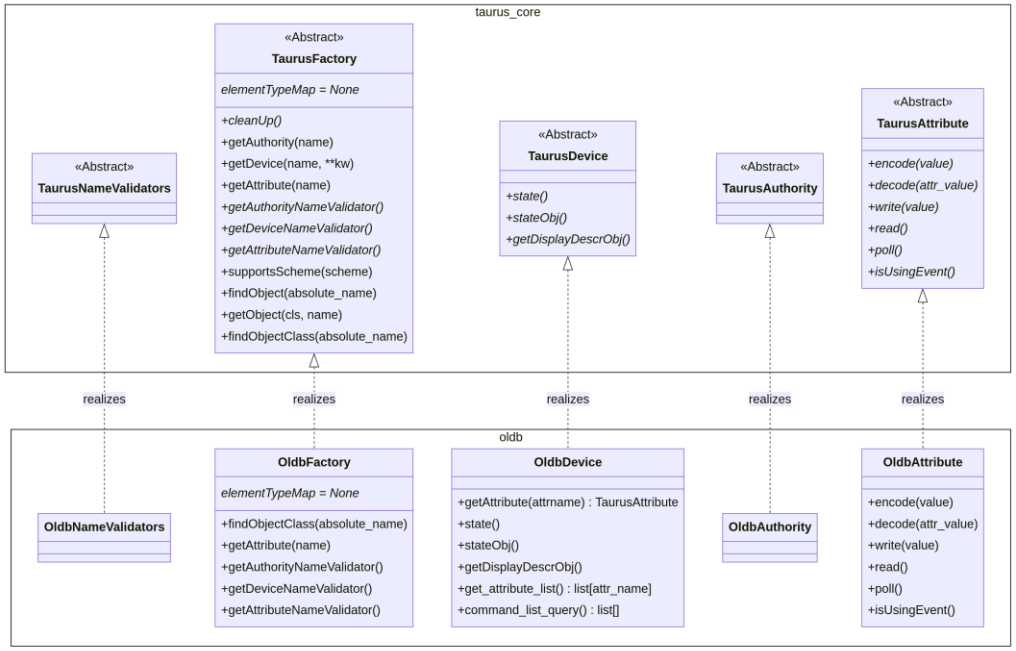


Figure 3. Class diagram of *taurus.core* model classes, their abstract methods indicated in *italic*, and their implementation in *cii.oldb* module.

Attribute requires the implementation of *read(), write(), encode(),* and *decode()* methods. It is possible to implement read-only models, such as h5file scheme plugin [7]; as did the first version of cii.oldb scheme plugin. These forego the *write()* and *encode()* implementations.

*read()* handles cache logic and performs a read (or similar) operation to the control system. The result is passed then to *decode(),* which act as a soft-barrier between the control system and taurus concepts. It translates from control system datatypes and available metadata to Taurus datatypes and metadata. With the data produces a *TaurusAttrValue* object which carries the value, units, datatype, format, and timestamp. *encode()* and *write()* perform the reverse operations.

Device list of methods to realize include *get_attribute_list()* and *command_list_query()*. These commands belong to the Tango API, and allow to conduct introspection on devices. Though not part of the Taurus interface, they are needed to make use of *TaurusCommandButton* and *TaurusDevicePanel* widgets. An improvement for Taurus will be proposed to make them part of the interfaces.

ELT's OLDB supports subscriptions to datapoints. Taurus does not have a particular method included in the interface of the *TaurusAttribute* class for this: Normally a TaurusAttribute when doing polling generates an event of *TaurusEventType.Periodic* type. When processing the callback from the OLDB we generate instead a *TaurusEventType.Change* event. It is encouraged to reuse *decode()* to obtain the needed *TaurusAttrValue* to send with the event notification. Care with access to cached value and it replacement operation is needed. –such is the case of OLDB— the callback is executed in a new thread. Mutex/Lock use to access the cached value is required.

Initial manual read is needed to avoid having no value until the next event is received. Taurus uses an observer/listener software design pattern in the *TaurusModel* to propagate these events. It includes special handling to only subscribe to events while there are listeners. Taking tango scheme plugin as example, the cii.oldb implements the same logic.

## CONCLUSIONS

One of the first GUIs developed was a Central Control System (CCS) prototype application, which used the first versions of the cii.oldb scheme plugin. At the beginning it supported only read-only operations through subscription. A bug in the subscription implementation of the OLDB middleware forced us to move to polling, which was easier to implement. It also allows us to continue implementation of other features while we investigated the issue. The plugin now has a more complete feature set, supporting read and write operations, changes over polling, subscriptions, metadata support, quality mapping, and warning ranges.

In figure 4 a recent version of the High Level Central Control (HLCC) Engineering GUI is shown. Of the widgets, several are in fact new taurus widgets:

- The Dartboard on the left is a taurus widget. It has two models, each a two value array of coordinates. They are used to update the Target and Current position of the telescope.
- On top of the Dartboard is the Telescope Global State widget. This widget uses taurus to do databinding and presents states from a statechart notation with different colors and quality background. It also allows to see a detailed representation of the state on click.
- The Monitor Docking widget on the right side is a custom widget that aggregates Taurus Widgets similar to what the TaurusForm widget does.

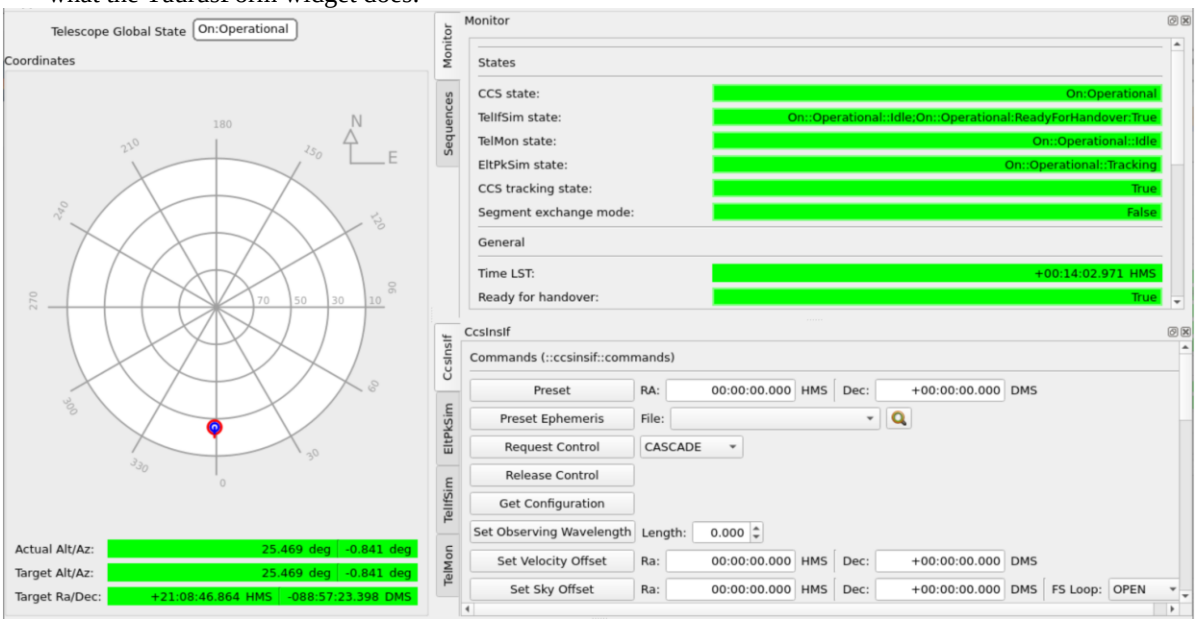


Figure 4. HLCC Engineering GUI to access multiple telescope monitor points from OLDB and RPC interfaces.

Two TaurusDevicePanel widgets are shown in Figure 5; they interact with the DemoService application through mal.rr plugin, and oldb plugin. The corresponding command lines are:

- taurus device zpb.rr://localhost:12801/DemoService?if=::demoserviceif::Commands
- taurus device cii.oldb:///cut/demoservice/instance1

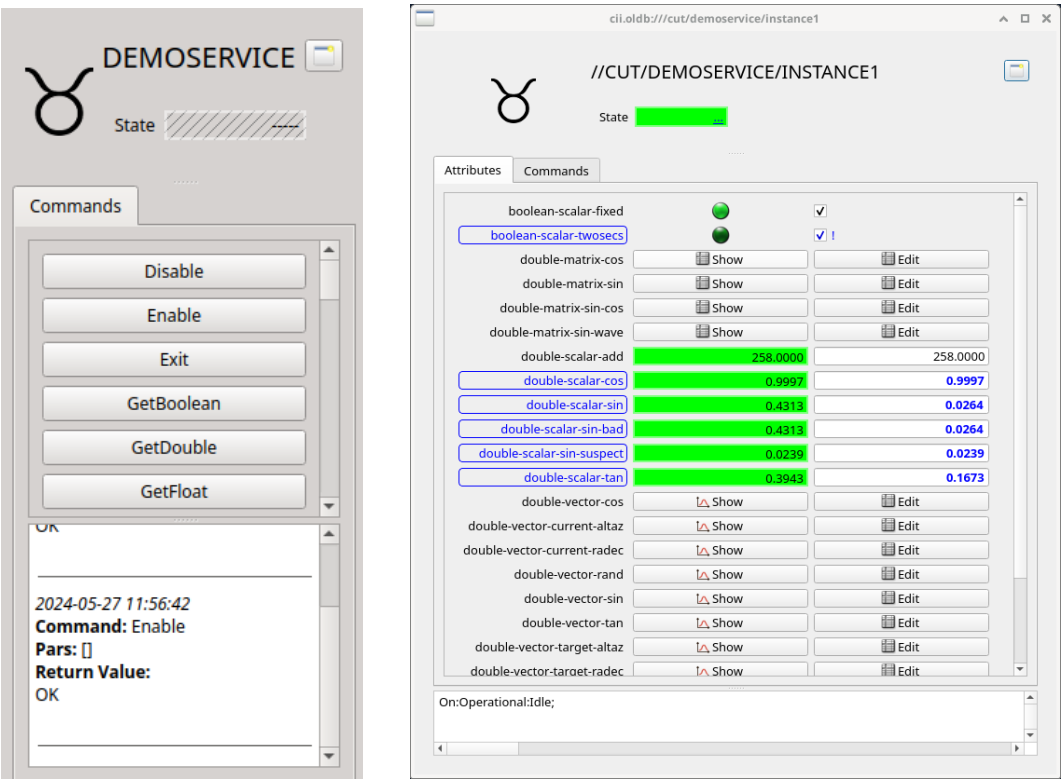


Figure 5. Left: *TaurusDevicePanel* for a MAL RR Demonstration Server application. Right: TaurusDevicePanel for an OLDB tree branch.

Declarative databinding is possible with the use of Taurus and plugins. Qt UI file declares the graphical design and through URIs models are specified. OLDB and MAL RR scheme plugins for Taurus make the implementation of GUIs in ELT much simpler with less code base to maintain.

1. Users indicate that declarative data binding is extremely convenient, as connection, mapping, polling, and subscription are all implemented.
2. Correct and consistent URI to Taurus Model mapping is crucial.
3. MVC is integral to Taurus. While keeping a well-defined interface, it allows heavy code reuse, and extensibility

Some future work includes:

- *malrr* scheme plugin allows only synchronous RPC calls. A standardization of asynchronous calls is to be proposed along with a *TaurusCommandButton* widget that performs such operations.
- Development of *TaurusModelChooser* plugin for cii.oldb is underway. Though not discussed in this manuscript, it is a Taurus extension point that provides navigation of available Devices and Attributes. Formalization of server application discovery mechanisms for ELT is undergoing specification. ELT is using Nomad to start, stop and track execution of server applications. Consul is used to store URIs, metadata and perform health checks. Once it is settled, the implementation of the *TaurusModelChooser* for mal.rr scheme plugin will be possible. You may learn more about Nomad [8] and Consul [9] in their references.
- Implementation of the third scheme plugin for MAL Publish Subscribe mechanism.
- Taurus Form Factories entry point plugin could be used to replace our custom replacement of the Taurus form seen on Figure 4.

## REFERENCES


[1] Taurus website, https://taurus-scada.org
[2] Tamai, R. et al., "ESO's ELT halfway through construction ", these proc. SPIE 13094, paper 13094-43 (2024)
[3] Pascual-Izarra, C., Cuní, G., Falcón-Torres, C., Fernández-Carreiras, D., Reszela, Z., & Rosanes, M. (2015). Effortless creation of control & data acquisition graphical user interfaces with taurus. *THHC3O03, ICALEPCS2015, Melbourne, Australia*.
[4] Chiozzi, G. et al., "Status of the ELT control software development", these proc. SPIE 13101, paper 13101-4 (2024)
[5] Scheme Plugin extension point, https://taurus-scada.org/devel/plugins.html#schemes
[6] Taurus Core Tutorial, https://taurus-scada.org/devel/core_tutorial.html
[7] H5file scheme plugin code, https://gitlab.com/taurus-org/h5file-scheme
[8] Nomad Introduction, https://developer.hashicorp.com/nomad/intro
[9] Consul Introduction, https://developer.hashicorp.com/consul/docs/intro