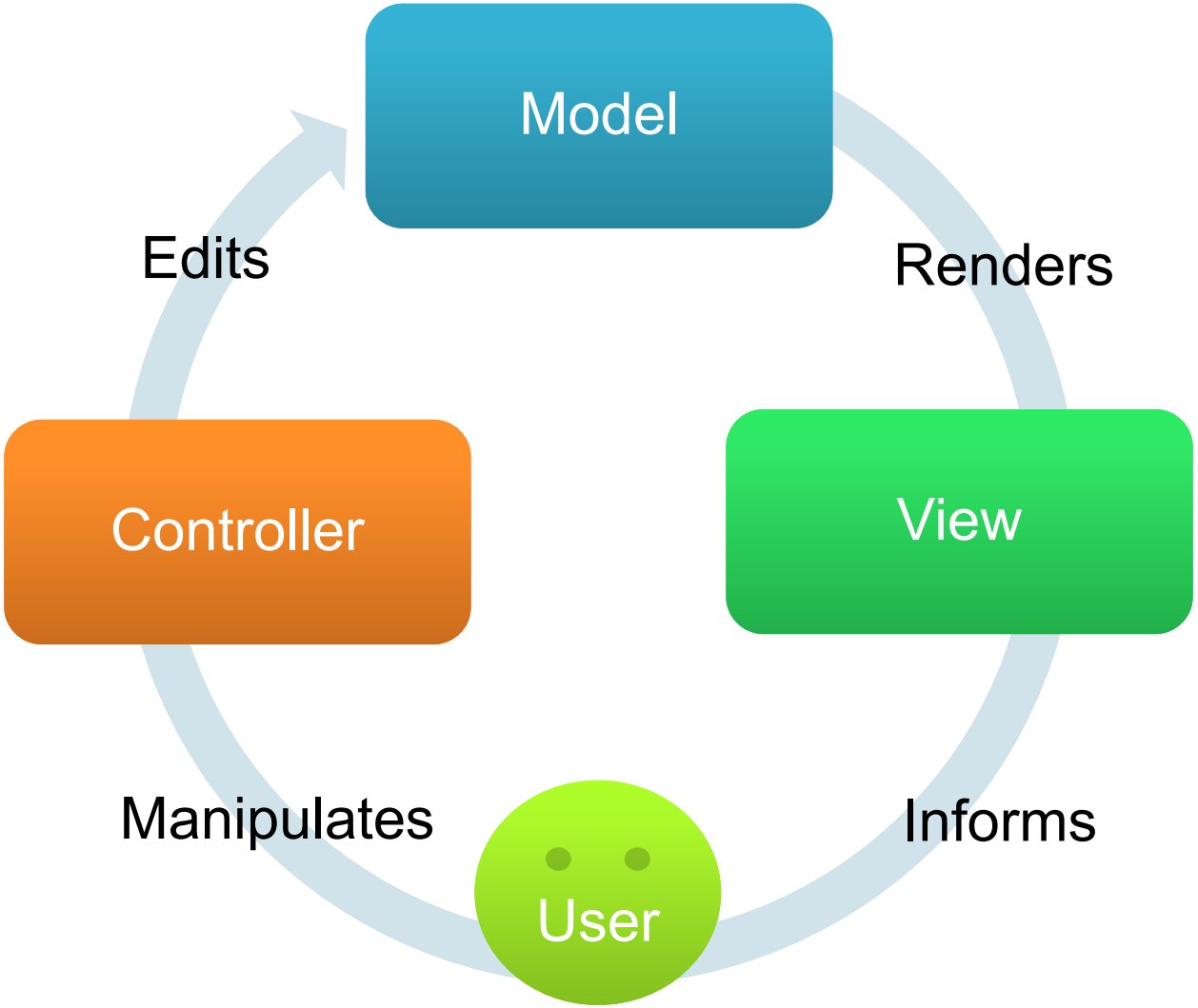
CCS UI Framework
Taurus Prototype

Arturo Hoffstadt<ahoffsta@eso.org>

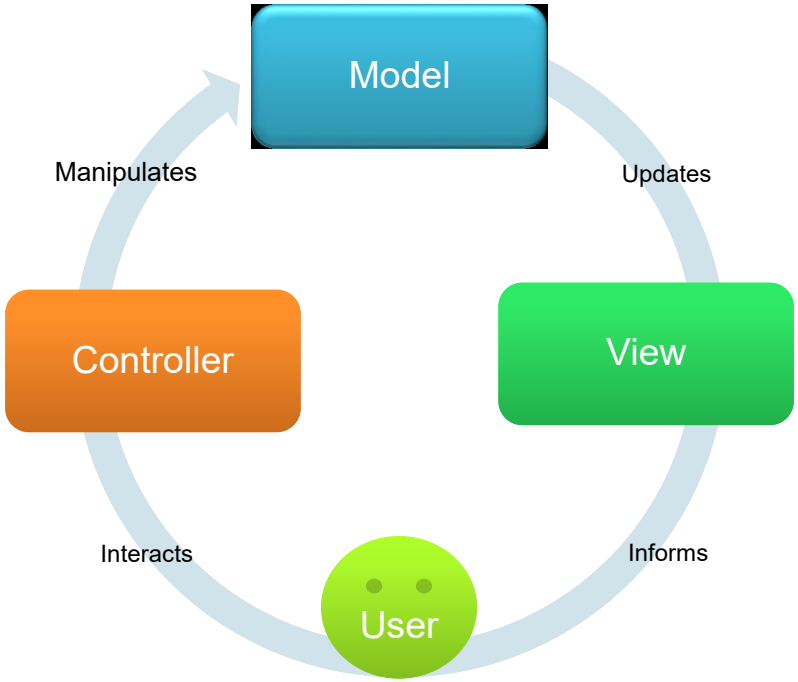
Short Introduction to MVC

Taurus MVC

MVC Overview

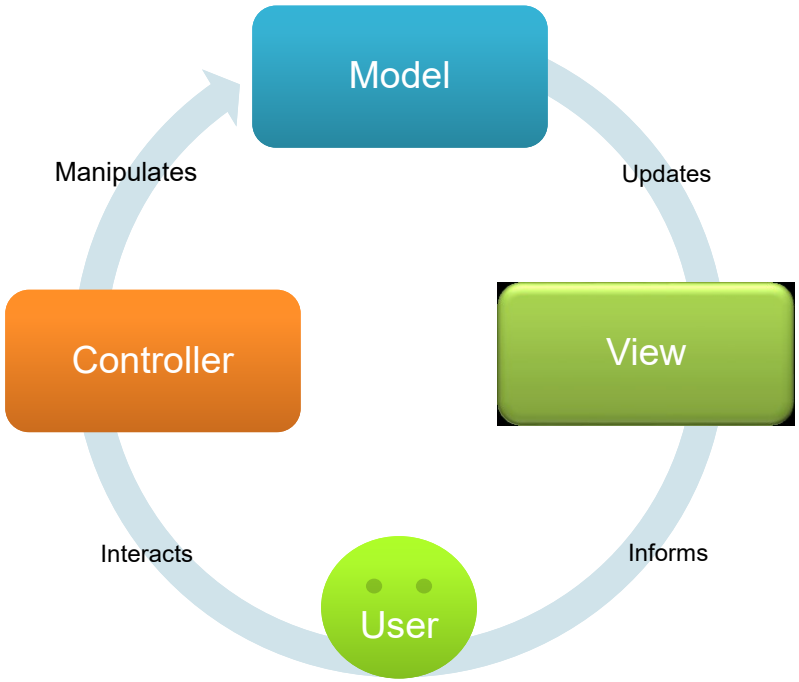


MVC Overview



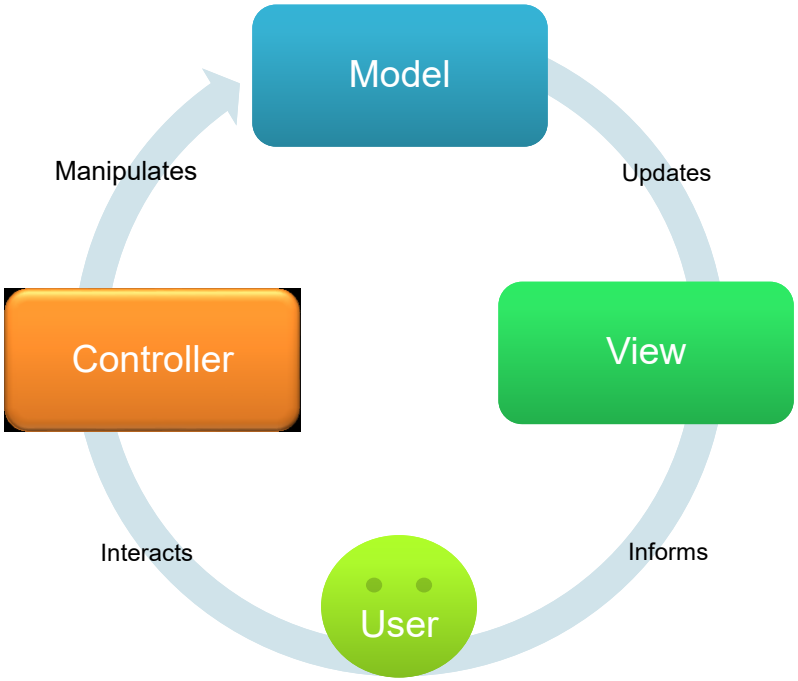
- The **Model** is a section of the data from the domain of the application.
- **Responsabilities:**
  - Contains the data
  - Knows how to read from its source
  - Knows how to write it back to its source
  - Translate any metadata into usable **Roles**

MVC Overview



- The **View** presents to the user the data from the model.
- **Responsabilities:**
  - Takes a subset of entries in the model, and presents it.
  - Determines the layout of the presentation (list, table, tree, heterogenous, etc)
  - Each piece of data can use a different Widget, these are called **Delegates**.

MVC Overview



- The **Controller** takes the inputs from the user, and makes the necessary changes to the model.
- Responsabilities:
  - Keeps references to Model and View.
  - Process input from the user, converting it to domain compatible notations.
  - Can also alter the user input.
  - Can manipulate the model, so it changes what is presented.

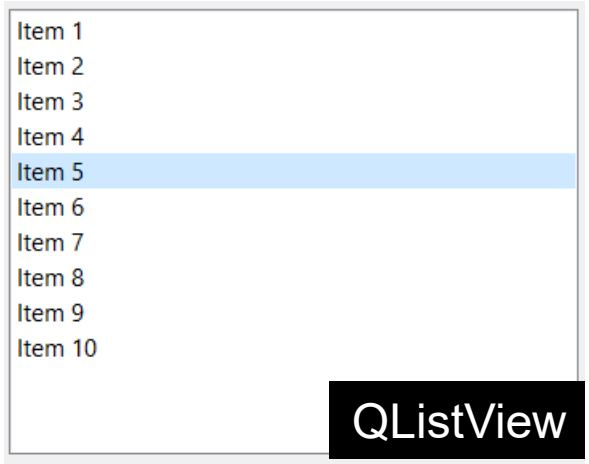
MVC in Qt

```
QStandardItemModel model;
for( int i = 0 ; i < 10; i ++){
     auto *item = new QStandardItem( QString("Item %0").arg(i+1) );
     this->model.setItem(i,0,item);
}
this->ui->listView->setModel(&this->model);
```

QList<QStandardItem>

The setModel() method in the view creates this connection

signal dataChanged() -> slot dataChanged()



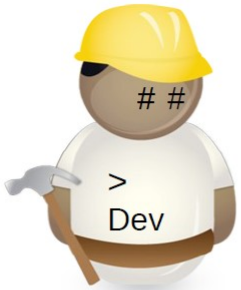
https://taurus-scada.org/

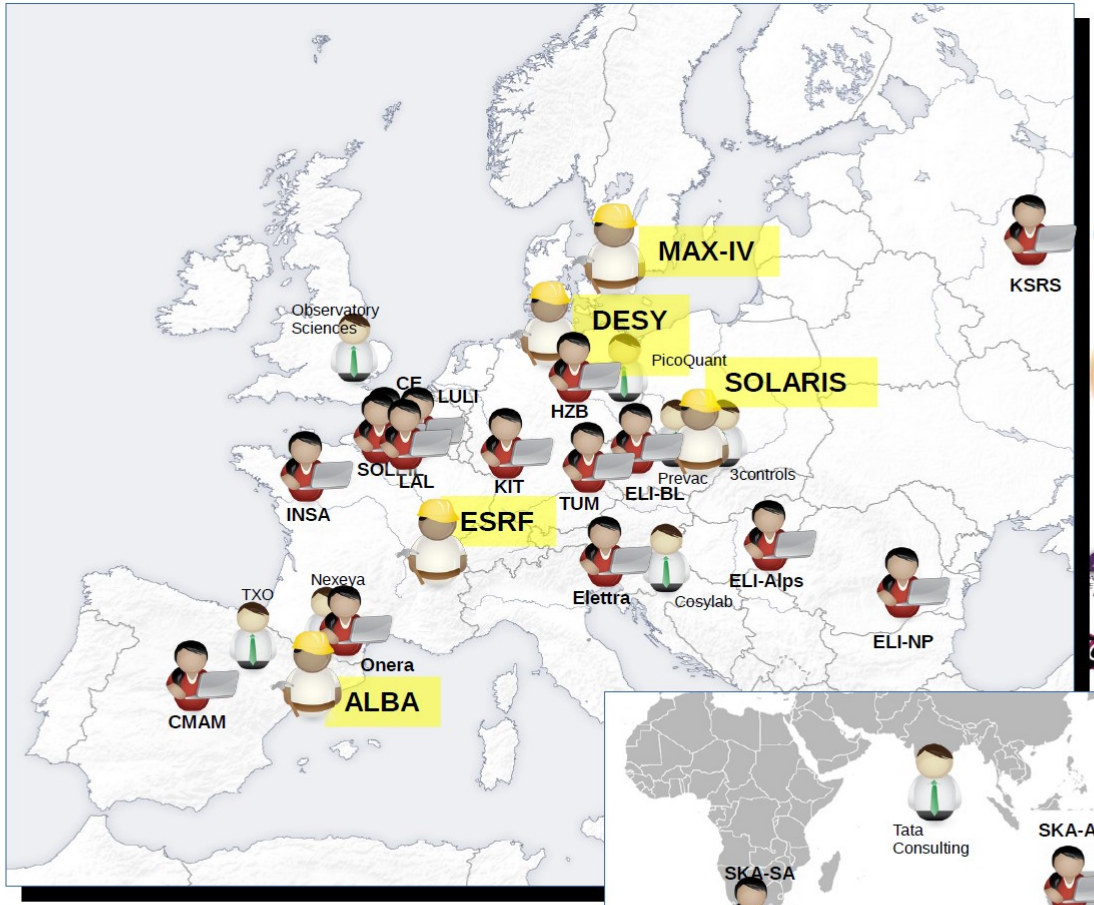
TAURUS

What is Taurus?

- Is a UI framework developed in python.
- Its based on MVC pattern.
- It is extended through "**scheme**" plugins. Each plugin provides connections to new protocols.
  - Tango, Epics, python evaluated functions, HDF files, pandas, CII OLDB
- It uses **URI** to identify datapoints or devices. A datamodel is defined by one or more URIs.
- OpenSource, LGPL V3

Taurus Community

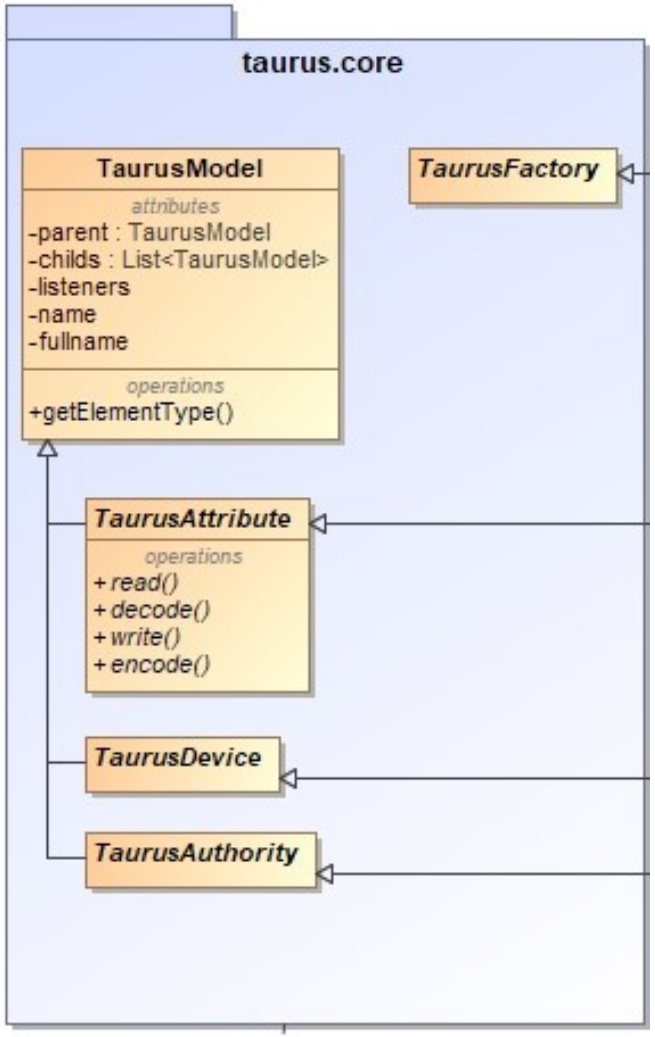




What does Taurus Framework provides

- CLI interfaces
- Several new QWidgets (LEDs, wheels, LCD)
- MVC for attributes and devices.
- TaurusWidgets
  - Common QWidgets that have model capabilities.
  - You can still use the common ones.
- Quick prototyping tools
- Application creation wizard

Taurus Model



- A set of 5 classes:
  - **TaurusModel**, the base class for all model elements.
  - **TaurusDevice**, which represents branches in a tree like structure.
  - **TaurusAttribute**, which represents leaves in this tree like structure.
- TaurusModel base class and TaurusDevice class implements a *Composition* pattern: Any device can have multiple children, but attributes cannot.

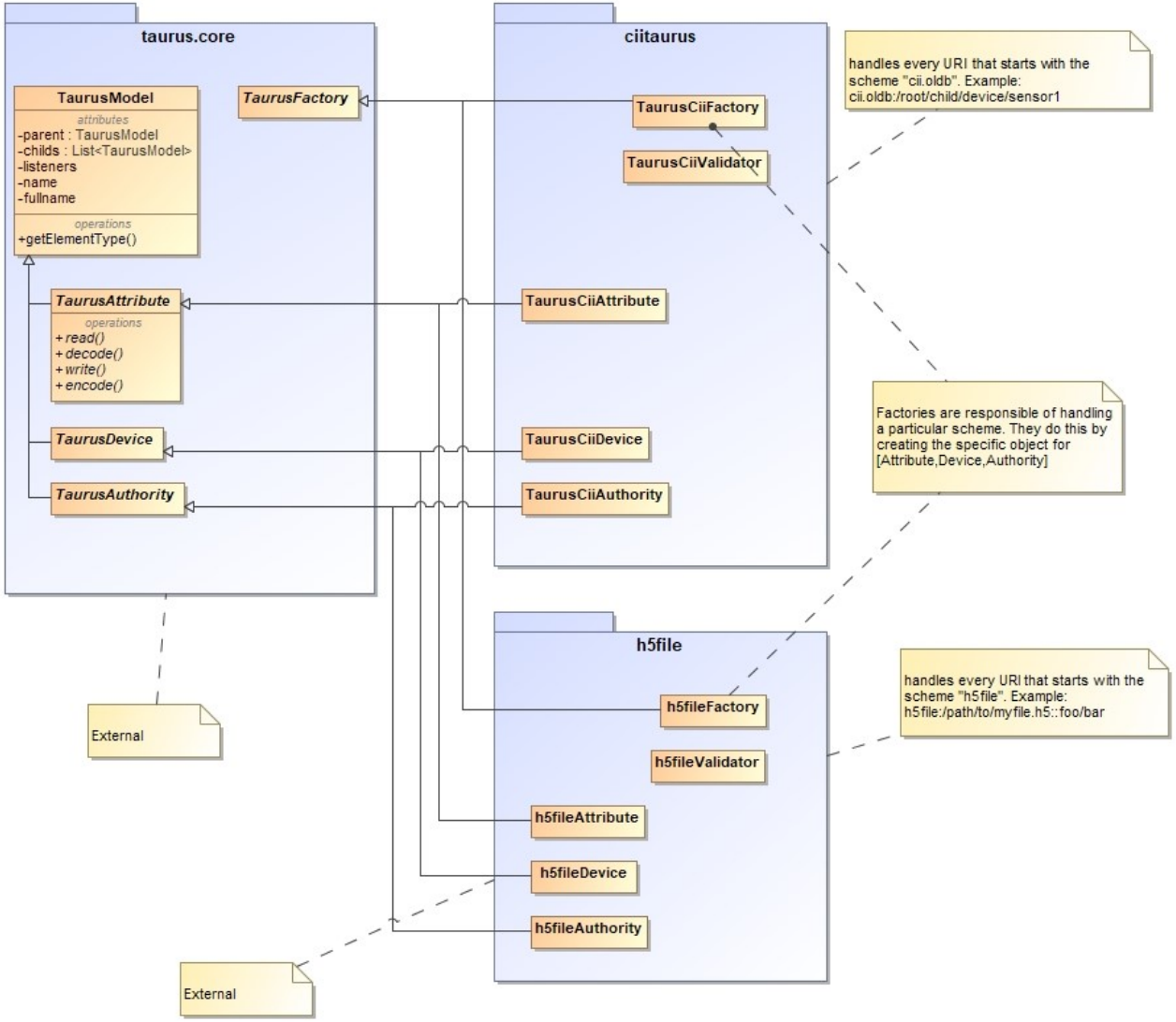
Taurus Attributes and Devices

- Example CII OLDB contents:

```
/root/
    example/
        double/
            sine                    <- datapoint
            cosine                  <- datapoint
            tan                     <- datapoint
        boolean/
            switch                  <- datapoint
```

- We map a CII OLDB datapoint to a TaurusAttribute:
  - `cii.oldb:/root/example/double/sine`
  - `cii.oldb:/root/example/switch/Boolean`
- A Device from the CII OLDB point of view could be:
  - `cii.oldb:/root/example/double`

Taurus Model



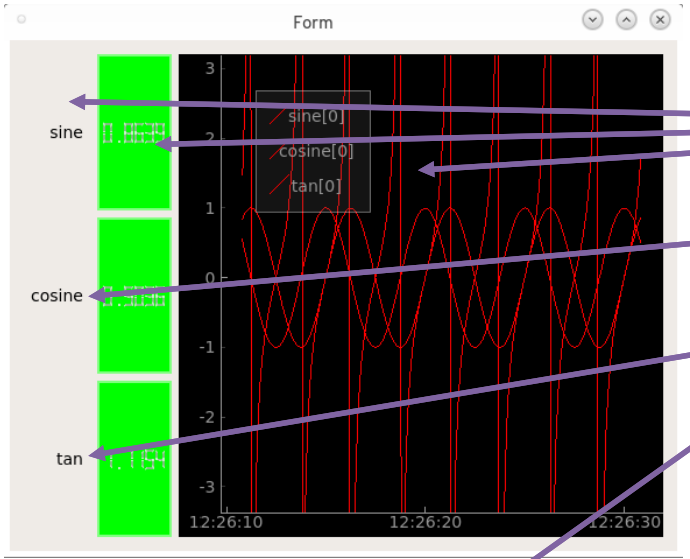
Taurus Model

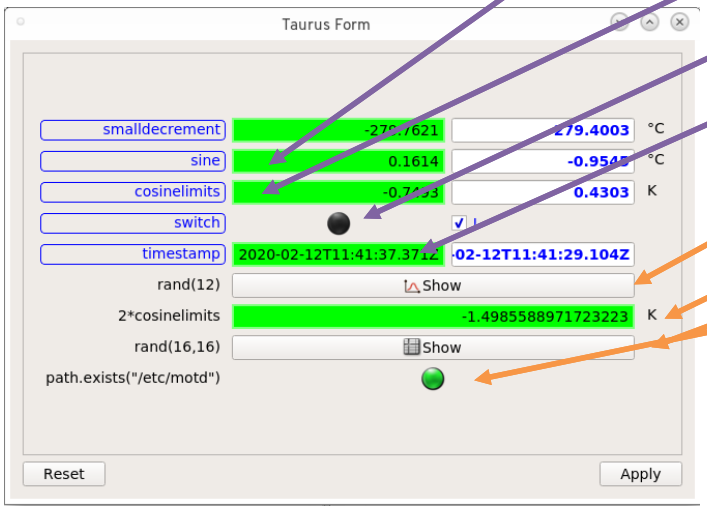
- The **TaurusFactory** in the model provides an *Abstract Factory* pattern, that will provide most of the logic to create the needed entities, but specifics are left to a particular scheme plugin.

```
import taurus

taurus.Attribute('cii.oldb:/root/example/double/sine')

taurus.Attribute('eval:rand(256)')
```

- Then the TaurusFactory will find out the scheme for the attribute.
- Using the matching plugin, it will request instances of the CiiFactory and CiiValidator.
- Will check the validity of the uri using the CiiValidator
- And the create the attribute using the CiiFactory.

Taurus Model





cii.oldb:/root/example/double/sine

cii.oldb:/root/example/double/cosine

cii.oldb:/root/example/double/tan

cii.oldb:/root/example/double/cosinelimits

cii.oldb:/root/example/boolean/switch

cii.oldb:/root/example/string/timestamp

eval:rand(12)

eval:2*{'cii.oldb:/root/example/double/cosinelimits'}

eval:rand(16,16)

eval:@os.*/path.exists("/etc/motd")

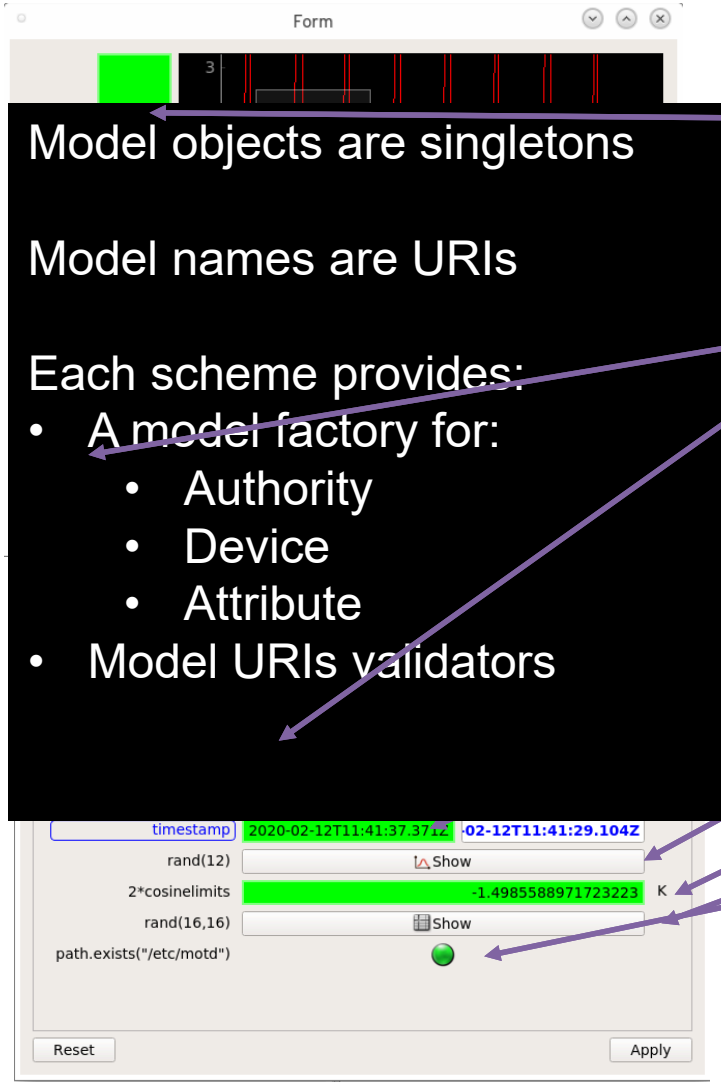
Taurus Model

Model objects are singletons

Model names are URIs

Each scheme provides:

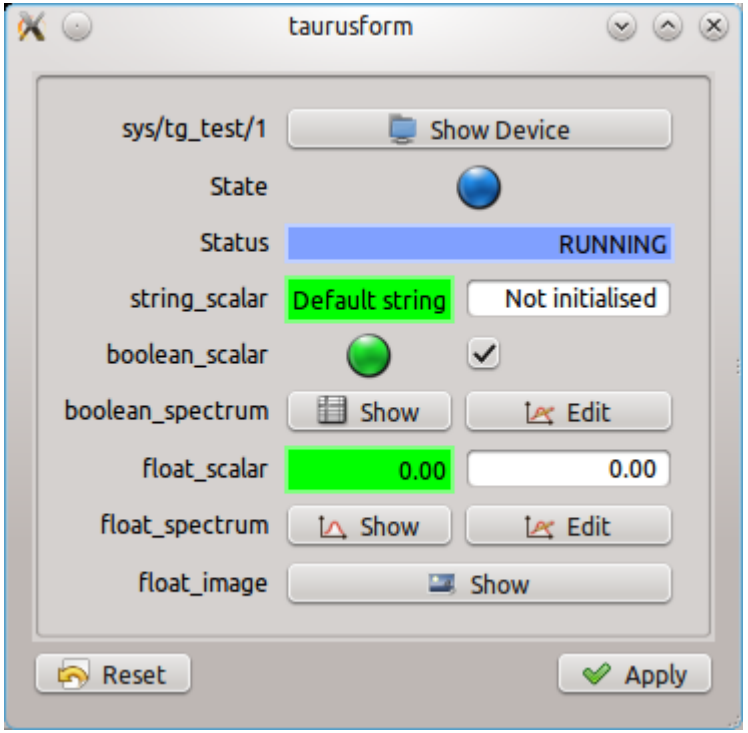
- A model factory for:
  - Authority
  - Device
  - Attribute
- Model URIs validators



- cii.oldb:/root/example/double/sine
- cii.oldb:/root/example/double/cosine
- cii.oldb:/root/example/double/tan
- cii.oldb:/root/example/double/cosinelimits
- cii.oldb:/root/example/boolean/switch
- cii.oldb:/root/example/string/timestamp
- eval:rand(12)
- eval:2*{'cii.oldb:/root/example/double/cosinelimits'}
- eval:rand(16,16)
- eval:@os.*/path.exists("/etc/motd")

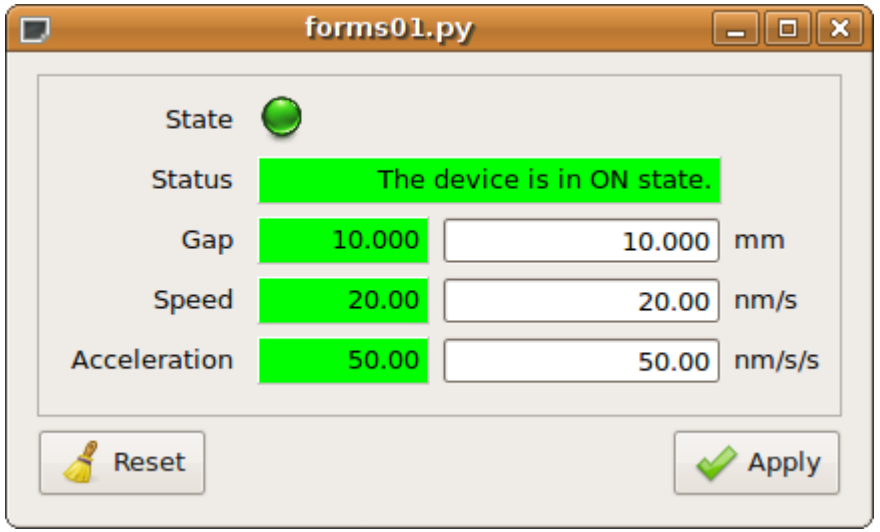
Taurus Views

- Taurus Widgets
- Taurus Form
- TaurusDevicePanel
- Array Editor
- Table View
- Plots, Trends



Taurus Controllers

- Each Basic QWidget implemented in Taurus has a controller.
- It transforms the value from the device into something meaningful.
- It also takes the user value and transforms it into the control system can use.



DEMO TIME!

Conclusions

- Model is reusable through several widgets.
- Extending Taurus is rather easy. 5 classes needs implementation, and can be done in a modular way.
- Model is not actually tied to Qt, you can use it for CLI programs as well.
- Quick prototype: bash line:
  - `$> taurus form 'cii.oldb:/…'`
- `Application Wizard:`
  - `$> taurus newgui`
- … to more complicated interfaces:
  - Design a .ui file in Qt Designer, with Taurus Widgets.

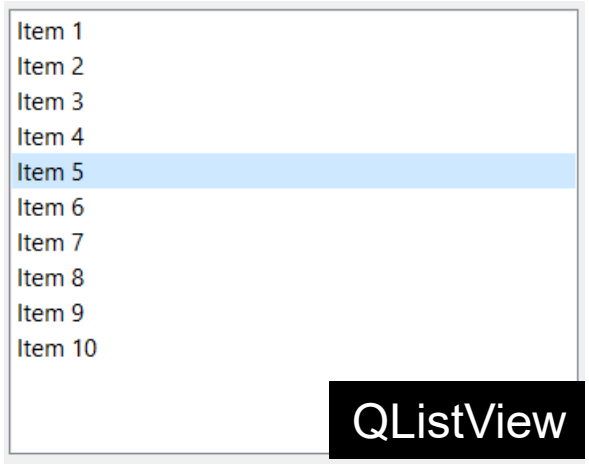
MVC IN QT

MVC in Qt

```
QStandardItemModel model;
for( int i = 0 ; i < 10; i ++){
     auto *item = new QStandardItem( QString("Item %0").arg(i+1) );
     this->model.setItem(i,0,item);
}
this->ui->listView->setModel(&this->model);
```

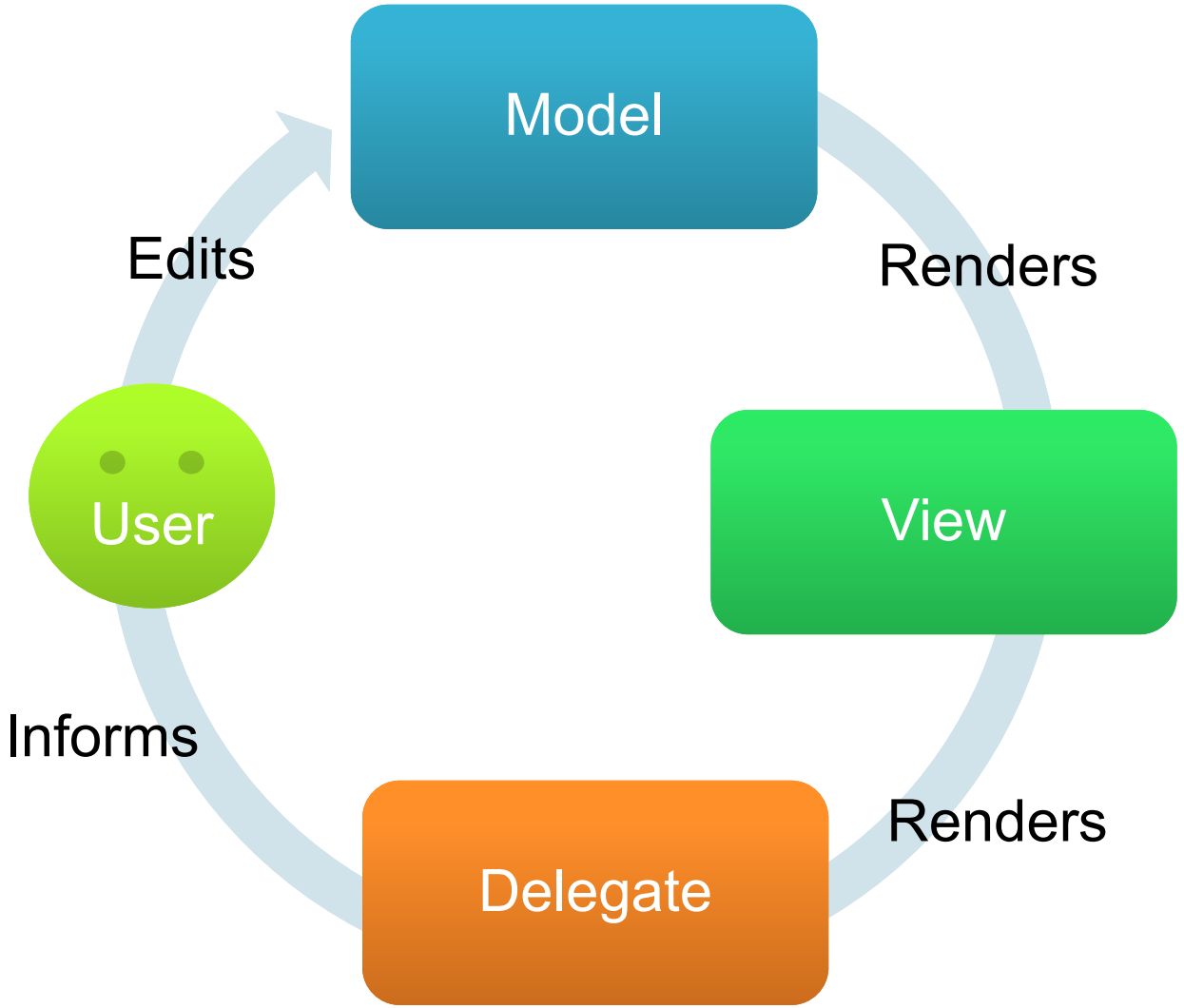
QList<QStandardItem>

signal dataChanged() -> slot dataChanged()



Where is the Controller?

Model-View framework in Qt



MVC in Qt

- Qt offers a series of ready to consume Views.
  - QListView
  - QTableView
  - QTreeView
  - QColumnView
- The controller is already implemented. The signal/slot mechanism makes the implementation of the controller trivial.
- Views implements the QAbstractItemView interface
- Models implements the QAbstractItemModel interface

MVC in Qt

1. Double click on an item
2. QListView will create a new instance of the proper Editor Delegate. In particular, the delegate for Strings -> QTextField.
3. We enter a new value, and press enter.
4. Entering a new value triggers a signal, valueChanged().
5. The View has already connected for us the signal for that particular delegate, against the model setData(index, value).
6. The model takes care of updating the value.
7. When the value in the model changes, a signal from the model is emitted, dataChanged()
8. All other views are connected to this signal. This is how each one of them also updates their own representation of the value changed.

MVC in Qt

- Role
  - Each item in the model can have many other metadata set.
  - We can customize appearance, or store other values.
  - Qt::DisplayRole: what is displayed in the Delegate
  - Qt::UserRole: an extra data container. For example, a complex type, or a int that represents the real value.
  - Qt::DecorationRole: An icon or color used while rendering the item.
  - Qt::StatusTipRole: String to present as tooltip on mouseover.
  - Qt::SizeHintRole: The size of the delegate.
  - Qt::BackgroundRole: Brush or color used to draw the background of the delegate.
  - Qt::ForegroundRole: Brush or color used to draw the foreground of the delegate.

MVC in Qt

- Delegate:
  - Widgets used to present an item.
  - The delegates normally uses the type of the DisplayRole to determine what widget to present.
  - For example: If DisplayRole has a Date, then a date selector is the Delegate.

MVC in Qt

- Role and Delegate customization:
- Qt suggests custom behavior to be programmed through Roles, and then through Delegates.
  - We can add new roles.
  - We can change the delegates.
  - We can create new roles or delegates.
- When full control of the items on the Model is needed, use the QAbstractItemModel.
  - You need to implement logic to get data from its source, and save it back.
  - You need to understand how the QModelIndex Works.

MVC IN TAURUS

MVC in Taurus

- Simple widgets are Views in Taurus as well:
  - TaurusLabel, TaurusLED, TaurusLCD, etc
  - These have an associated **model.**
  - The Widget is just responsible for storing “configuration” properties (model, which role is going to be used for presentation)
- The Controller for each one of those widgets:
  - TaurusLabelController, TaurusLEDController, TaurusLCDController, etc
  - The Controller is in charge of handling the attribute and its metadata, to present it in the correct manner.
  - The Controllers uses the model roles for this.

MVC in Taurus

```
class TaurusLabelController(TaurusBaseController):
  def _updateForeground(self, label):
    fgRole, value = label.fgRole, ‘’

    # handle special cases (that are not covered with fragment)
    if fgRole.lower() == 'state’:
      value = self.state().name
    elif fgRole.lower() in ('', 'none’):
      pass
    else:
      value = label.getDisplayValue(fragmentName=fgRole)
    self._text = text = label.prefixText + value + label.suffixText
…

  def _updateToolTip(self, label):
    if not label.getAutoTooltip():
      return
    toolTip = label.getFormatedToolTip()
    if self._trimmedText:
      toolTip = u"<p><b>Value:</b> %s</p><hr>%s" % (self._text, toolTip)
    label.setToolTip(toolTip)
```

MVC in Taurus

```
class TaurusLabel(Qt.QLabel, TaurusBaseWidget):
  DefaultPrefix = ''
  DefaultSuffix = ''
  DefaultBgRole = 'quality'
  DefaultFgRole = 'rvalue’

  def _calculate_controller_class(self):
    ctrl_map = _CONTROLLER_MAP
    ctrl_klass = ctrl_map.get(model_type, TaurusLabelController)
    return ctrl_klass

  def resizeEvent(self, event):
     if not getattr(self, '_inResize', False):
       self._inResize = True
       self.controllerUpdate()
       self._inResize = False
     Qt.QLabel.resizeEvent(self, event)

  def setModel(self, m):
    self._controller = None
    self._permanentText = None
    TaurusBaseWidget.setModel(self, m)
    if self.modelFragmentName:
      self.setFgRole(self.modelFragmentName)
```

Taurus Controllers

- TaurusReadWriteSwitcher:
  - Controller that allow to present a read widget, an on doubleclick, a write widget.

Higher Level Models

- With access to a TaurusAuthority, developers can requests the following models
  - All these models inherits from TaurusDbBaseModel, which is a QAbstractItemModel
  - TaurusDbSimpleDeviceModel
    - A Qt model that structures device elements in 1 level tree with device name as node leafs. This model contains only 1 column
  - TaurusDbPlainDeviceModel
    - A Qt model that structures device elements in 1 level tree. Device nodes will have attribute child nodes if the device is running.
  - TaurusDbDeviceProxyModel
    - Filter & sorting facility for taurus models

Higher Level Models

- For the models from the previous slide, Taurus offers:
  - TaurusDbTreeView
  - TaurusDbTableView
- Auxiliary models:
  - QConfigEditorModel

Taurus – Other features

- BaseConfigurableClass
  - Allows to indicate a widget should persist its configuration.
- Taurus.qt.qtcore.model
  - creates qt pure models from TaurusModels.
- TaurusGrid
  - Instead of presenting the models as a form, presents them in a grid.
- Logging
  - uses python logging module. Every class is a logger.
- Factory Class registratations
  - It is possible to override the default class used to represent a device or attribute.

Taurus in detail

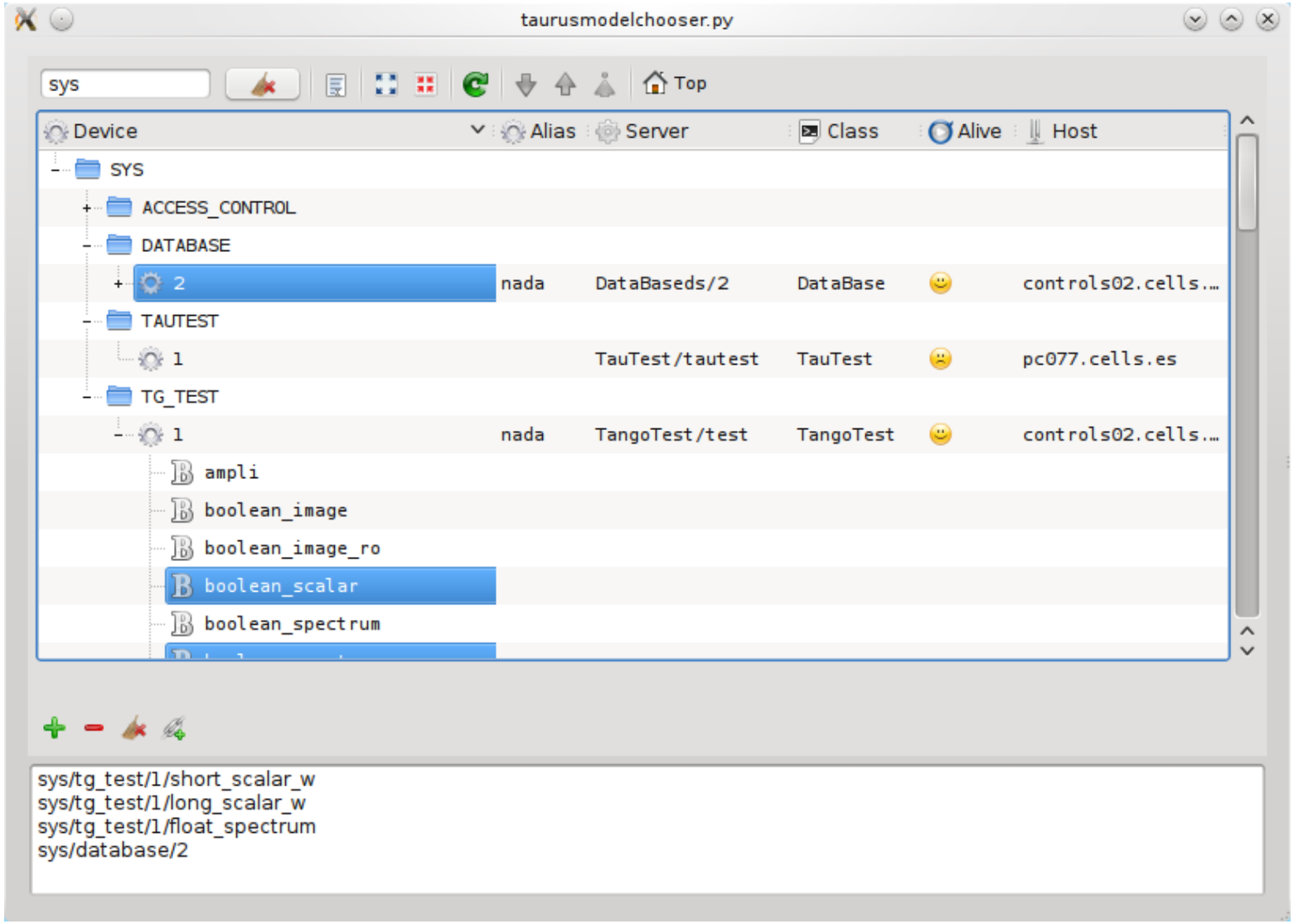
- TaurusWidget, TaurusBaseWidget
  - Base class for all widgets, introduces new properties (models, quality, modifiable)
  - Base class for all widgets
  - Drag and Drop handling
  - Configuration of widgets persistence
- TaurusBaseWritableWidget
  - Base class for all input widgets
  - Dangerous operations
- TaurusBaseController
  - Indicates how the values are presented form the TaurusAttribute to the widget, and written back. Also, the map between quality/state and background.
- TaurusMainWindow
  - MainWindow with support for fullscren, settings, perspectives and other features. Customizable on run-time or by wizard form.

FUTURE WORK

Future Work

- Implement Device and Authority in cii.oldb scheme
  - Will allow to have the tree like nature of the database directly represented.
- Implement cii.mal scheme
  - Is a multi-protocol scheme (reply-request and subscription)
  - IMO, the hardest, due to the safe-typed nature of the interface.
    - Will implement one prototype for one particular IF
    - Then, will implement a generic one
    - At some point, a map between URIs and IFs will be needed.

Taurus Authority



Future Work

- Customization of Views
  - The requirement in INS widget library documents specify views and layouts that are not the same ones as the defaults of Taurus.
- Deployment:
  - To use newer version of Qt, we need a new version of python. This will take some time to integrate well into the devenv.
- Bugs:
  - During a long running session, events from REDIS can me missed. This needs further research.
  - Changing the value in the write widget triggers 4 checks on the data value. Ideally, there should only be one.

Taurus Resources (just a few)

- Wiki:
  - https://github.com/taurus-org/taurus/wiki
- Best Practices:
  - https://github.com/taurus-org/taurus/wiki/Best-Practices-for-Taurus-4
- Status of Taurus presentation:
  - https://github.com/taurus-org/taurus/wiki/SardanaAndTaurusStatus-TangoMeeting2018.pdf
- Introduction to Taurus:
  - (conference video + slides + paper)
- Tutorials:
  - https://github.com/sardana-org/sardana-followup/blob/master/20180605-Prague/AGENDA.md