PAUL SCHERRER INSTITUT

PSI

An EPICS Display manager with Qt

Wir schaffen Wissen – heute für morgen

The EPICS display manager MEDM is a very powerful tool with many features and satisfies most of the needs for synoptic displays. However MEDM is based on MOTIF and X11, systems/libraries that are getting into age. Moreover applications using those systems are difficult to maintain and to extend. Today a candidate as a platform for new applications could be QT instead of MOTIF and some applications have already been written at PSI giving some experience with Qt. The step to try to use Qt as a base for a new EPICS display manager was therefore decided and a new application caQtdm (channel access Qt display manager) was started. In order to test the display manager, all the widgets and features existing in MEDM have to be tested. Moreover the thousands of MEDM data files (.adl) have to be used as a start for a new system. As of today the new manager integrates already most of the features of MEDM and a parser for translating the .adl files into Qt-Designer files is operational. The state of the project will be presented.

Outline of the presentation

- Motivation for “yet another epics display manager”.
- Qt: a new system for writing graphic user interfaces.
- Qt: Architecture of MEDM and caQtDM
- Qt-Designer: what can it do and what can’t be done.
- Qt: integrating widgets, code example
- Translator: parsing of .adl files.
- Some problems encountered underway
- Some display examples.
- Conclusion.
- Offline Demo.

yet another epics display manager: motivation

MEDM, very powerful, lots of features, widely used for all synoptic displays at PSI, however:

1. No more support, no new features

2. Based on MOTIF/Xt/X11 (for linux still ok, for windows reflectionX or other X11-server needed).

3. Many other display managers exist like css, jddd, …(not everybody wants to turn to java and integrated tools)

But what about all the thousands of MEDM description files (.adl) out there ?

→ Combine the writing of a new tool using a modern system (Qt) with the translation of the description files into terms of that tool.

Qt: a new system for writing graphic user interfaces

What is Qt (***definition given by Wiki***)**:**

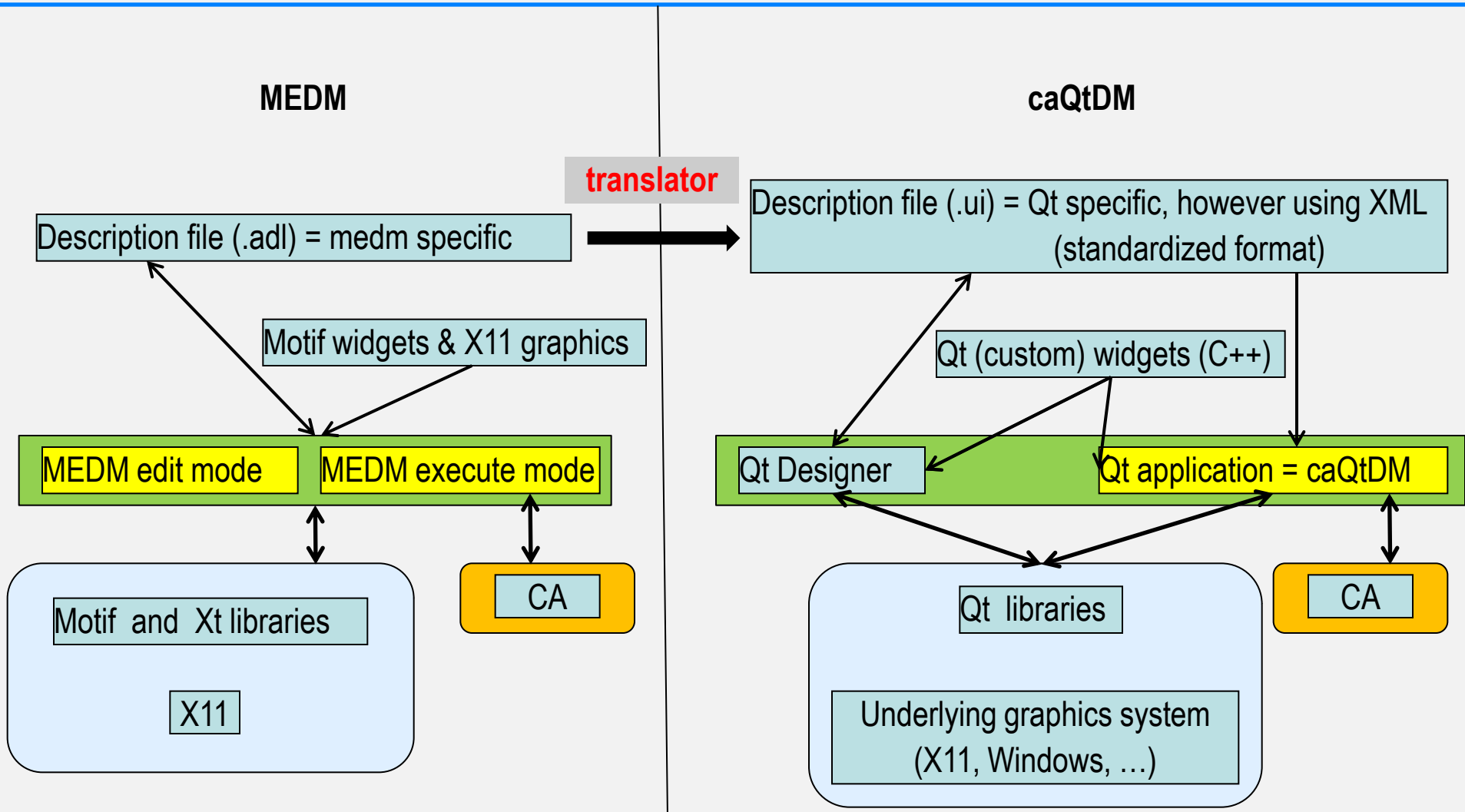
Qt is a cross-platform application framework that is widely used for developing application software with a graphical user interface (GUI) (in which cases Qt is classified as a *widget toolkit*

Qt uses standard C++ but makes extensive use of a special code generator (called the *Meta Object Compiler*, or *moc*) together with several macros to enrich the language. It runs on the major desktop platforms and some of the mobile platforms.

Qt comes with libraries, an integrated development environment (IDE) called QtCreator with an user interface designer (Designer) and some third party libraries like qwt for 2D graphics and qwtplot3d for 3D graphics.

The actual version used is QtCreator 2.2.0 based on Qt 4.7.3

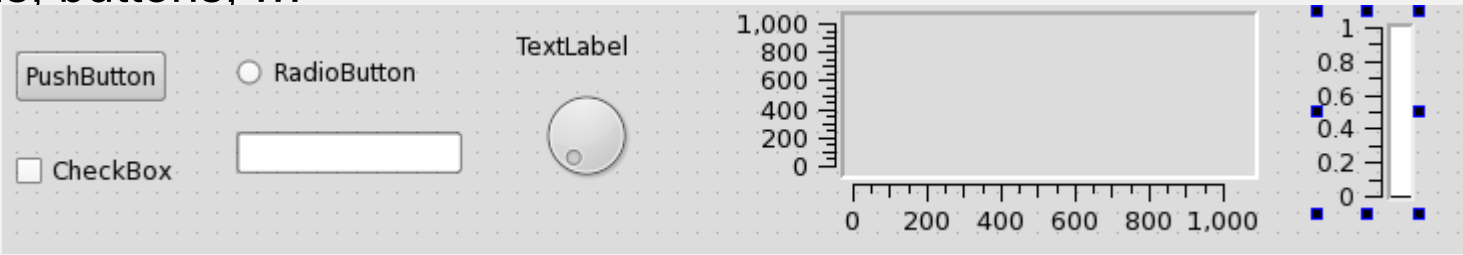
Architecture of MEDM and caQtDM



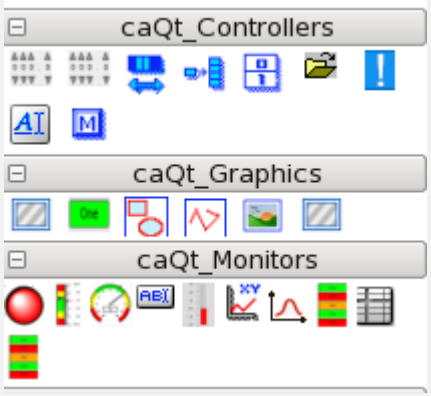
It has to be noted, that the widgets are not directly epics aware, but have properties and methods used by caQtDM

Qt-Designer, what can it do and what can't be done

- a tool for building graphic user interfaces with widgets like sliders, knobs, labels, buttons, …



- **and you can introduce your own widgets:** stripchart, cartesian plot, composite file, related display, …



- **Not** a drawing tool with lines, circles, arcs, … Designer gives no access to mouse events inside its widgets (or I missed that), solution:
  - simple drawing widget for rectangles, circles, ovals, lines can be created.
  - Polylines can only be created by introducing coordinates. (Translator will do that of course)

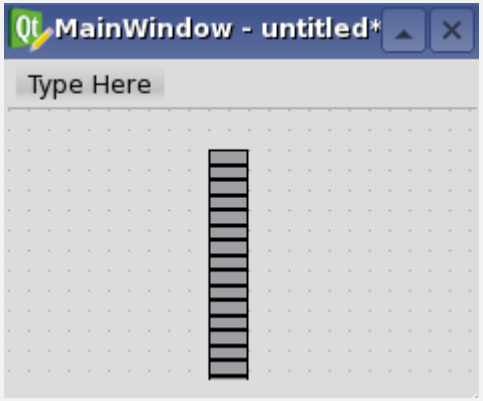
Qt designer and Qt applications: integrating new widgets

C++ framework + *Meta Object Compiler* + macros give following possibilities:

- **Inheritance:**
  - Use an existing widget class and add your own methods to it.
  - Use a Qt base class like QWidget or QFrame and add your methods like painting (drawing), events, …
- **Properties:**
  - Enrich your widget with the properties, associated methods and enums you need.
- **Plugin:**
  - Possibility to integrate your own widgets into the Qt-designer by describing them into a plugin (a .so library in linux).
  - Build the plugin against the widget library you created

Actually for caQtdm 4 libraries: **3** plugins (for controllers, monitors and graphics) and **1** library with all the custom widgets.

Qt-Designer: example of code integration



➢**Code example:**

```
class caByte : public QWidget {
Q_OBJECT
Q_ENUMS(Direction)
Q_PROPERTY(QString channel getPV WRITE setPV)
Q_PROPERTY(Direction direction READ getDirection WRITE setDirection

public:
        enum Direction {Up, Down, Left, Right};
        QString getPV() const;
        void setPV(QString const &newPV);
        Direction getDirection() const;
        void setDirection(Direction direction);
}
```

caByte	
channel	
direction	Down
startBit	0
endBit	15
foreground	[0, 0, 255] (255)
background	[160, 160, 164] (255)

The .adl files: translation

Translator (for now still called myParser) is based on parser code of MEDM and generates ui files.

Two modes exist:

1. Parses all composite files from a main .adl file and produces a flat file:
   - Advantage: everything can be seen in Qt-Designer
   - Disadvantage: one looses the power of “include files with macro substitution”

2. Parses the .adl file and produces includes:
   - Advantage: keep the power of include files with macro substitution, so that one change of the composite file will be reflected in all displays.
   - Disadvantage: one can not see directly the include in Qt-Designer, one has to load also the corresponding “composite” ui file.

Stylesheets

Qt uses stylesheets at different levels:

- At application level. caQtDM will load a specified stylesheet where color and font definitions can be defined.

- At ui file level. Every ui file can also contain a stylesheet definition.

- At widget level. Normally not used, while used for coloring with the widget methods

Fixing the font size in the stylesheet prevents however from redefining it later?

Some problems encountered underway

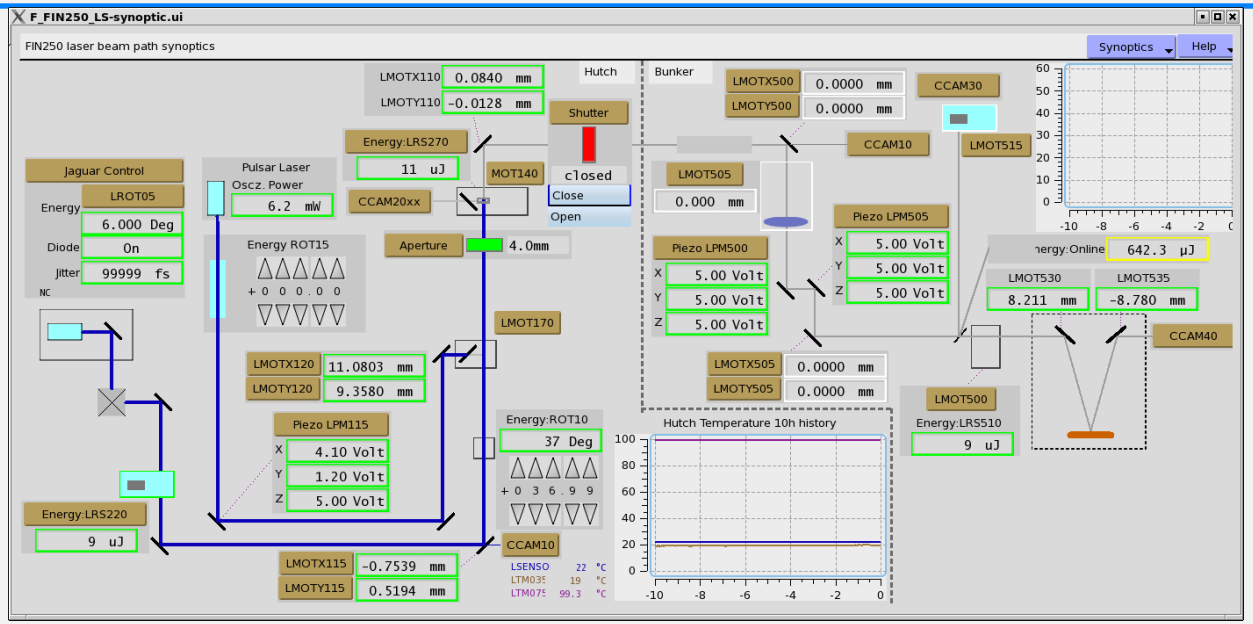
1. The widget ordering inside the MEDM description file is not the only ordering mechanism of MEDM. You need some digging inside the MEDM code to understand (more or less) the ordering giving the layering mechanism (some times the layering has to be adjusted with the designer).

2. In Qt, the width of a text widget has to fit the text, but in MEDM the width was not always adjusted (at least not by me). The translator will try to adjust the width (but not when macros are used)

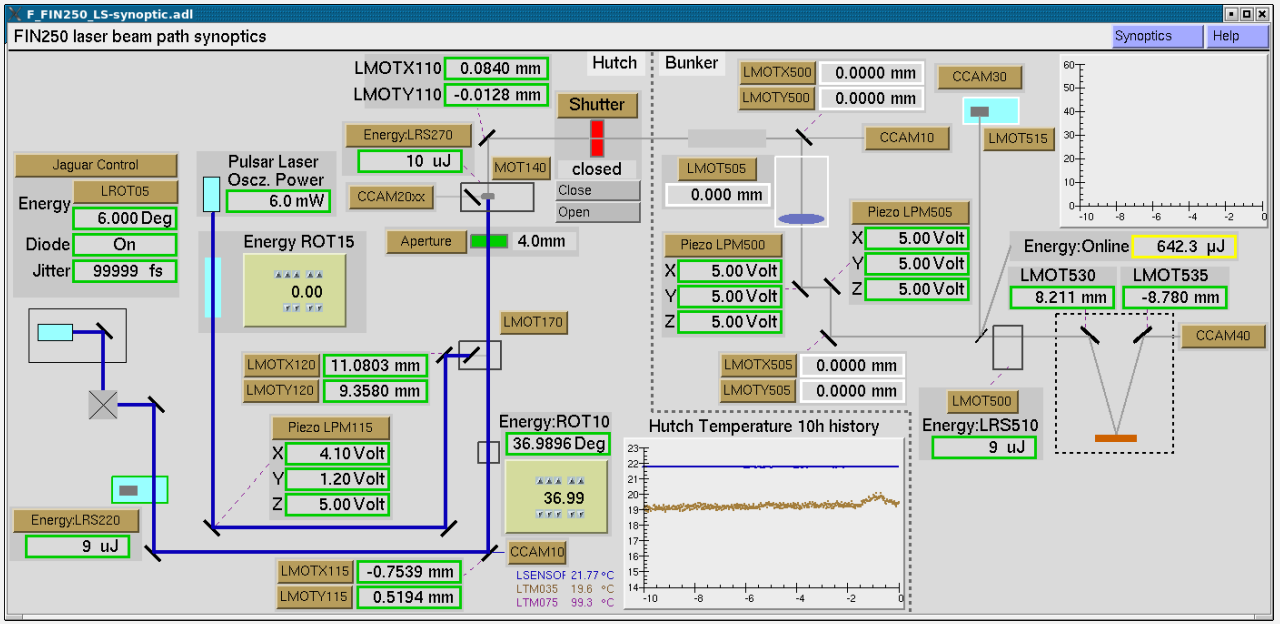
3. In Qt-Designer there are no drawing possibilities. Polylines can therefore not be drawn interactively while no access to mouse positions inside widget (we have to use coordinates introduced by hand). Perhaps I did not find the way to do it ?

Probably I will encounter more problems, when starting to parse more screens, while many people use MEDM differently in a more or less complicated way. At now it seems quite robust and produces in 99% (?) of the time correct files.

Example:



caQtDM



medm

Conclusion

- the .adl translator produces correct .ui files. Some few files have to be edited after translation when the layering is incorrect.
- caQtDM reproduces correctly the behaviour of MEDM. However some «bugs» will still be found and some minor used features (at PSI) have still to be integrated (ex. trigger channel for cartesian plot)
- More widgets are easily integrated. Actually table and led widgets have been added.
- Evaluate QtScript for scripting?
- The Qt designer presents a good editor tool. At PSI we have to start with some users.
- If any interest, I can give a demo.

Acknowledgments:

The authors of MEDM for their code.
Elettra people for providing some widgets.

- Thank you for your attention