Smart-appliance development with Qt

Nicola Valpiani 24/05/2018

Electrolux

Electrolux in brief

Electrolux shapes living for the better by reinventing taste, care and wellbeing experiences, making life more enjoyable and sustainable for millions of people.

As a leading global appliance company, we place the consumer at the heart of everything we do. Through our brands, including Electrolux, AEG, Anova, Frigidaire, Westinghouse and Zanussi, we sell more than 60 million household and professional products in more than 150 markets every year.

Electrolux has been doing business since 1919. The headquarters are located in Stockholm, Sweden, and the Electrolux share ELUXb is listed on Nasdaq OMX Stockholm.

Taste

Care

Wellbeing

Electrolux offering

Electrolux products include refrigerators, dishwashers, washing machines, cookers, vacuum cleaners, air conditioners and small domestic appliances. The Group is the only appliance manufacturer in the world to offer complete solutions for both consumers and professional

Cookers, Ovens and Hobs

Vacuum cleaners and Small appliances

Refrigerators and Freezers

Air Conditioners and Dehumidifiers

Dishwashers

Laundry products

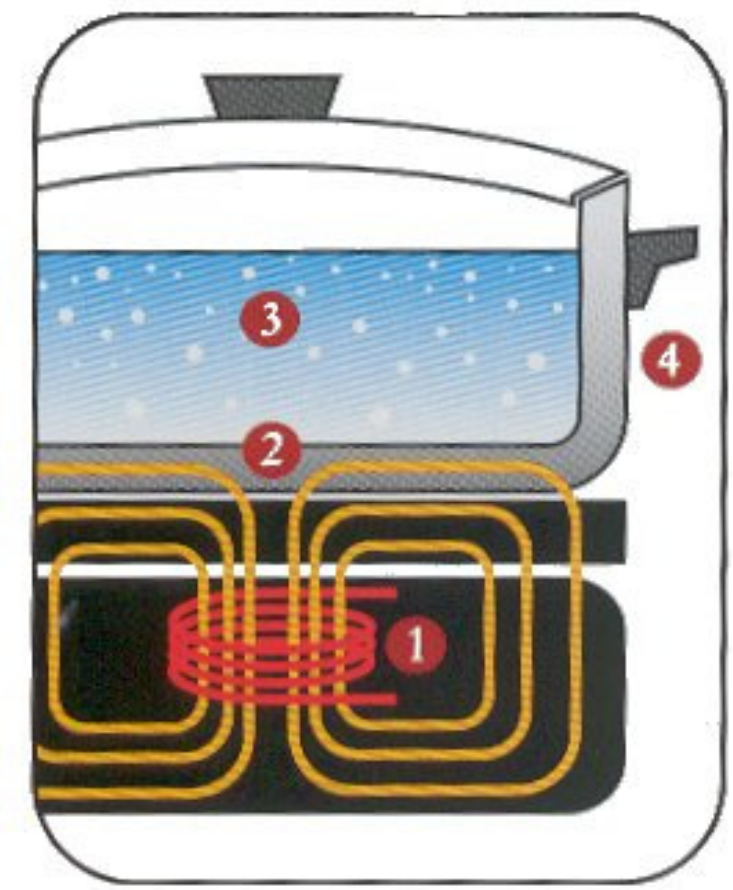
Products for professional use

Product description

With induction cooking you save time and a lot more

The Induction hobs combine the speed of gas and electricity with unbeatable comfort of handling. Meals are finished faster, because the selected power is available immediately and without any delay when switching on the zone. Our induction hobs also recognize whether or not a pan is placed on the cooking zone and what size it is, the direct energy transference makes it possible and this is why you do not have to match the size of the pan to the size of the zone. Only the bottom of the pan is heated and not the total area of the cooking zone.

Operation principle of induction



How Induction Cooking Works:

1. The element's electronics power a coil that produces a high-frequency electromagnetic field.
2. The field penetrates the metal of the ferrous (magnetic-material) cooking vessel and sets up a circulating electric current, which generates heat (Eddy currents).
3. The heat generated *in the cooking vessel* is transferred to the vessel's contents.
4. Nothing outside the vessel is affected by the field--as soon as the vessel is removed from the element, or the element turned off, heat generation stops.

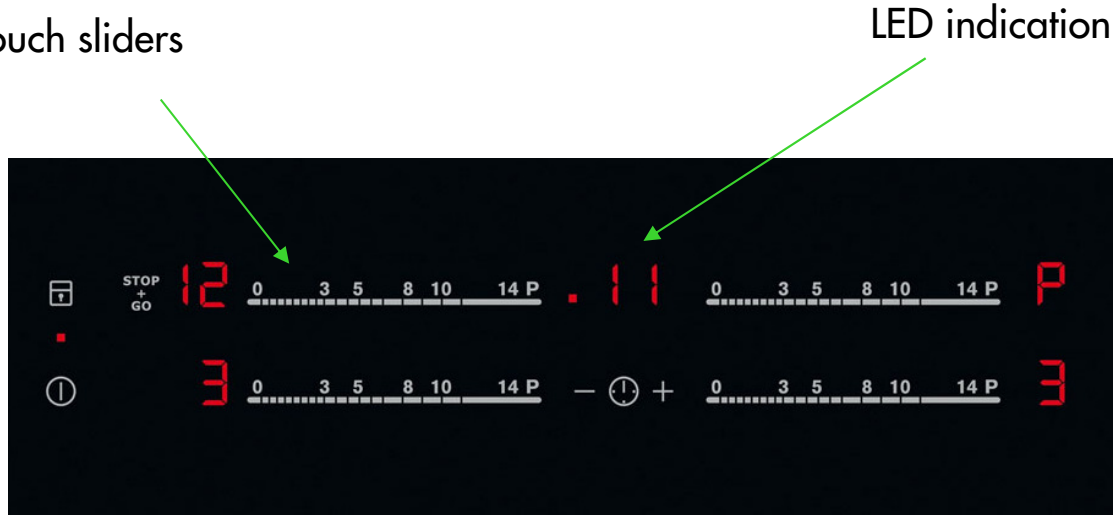
Source: theinductionsite.com/

Existing induction hobs

Example of existing induction hob

Examples of existing user interfaces

State of the Art LED



This kind of solution can't provide too much interaction due to limitation in terms of:

- Fixed touch keyboard
- Fixed LED displays and icons position and size

State of the Art TFT user interfaces

This setup will allow much more user interaction and customization, thanks to the TFT graphic

The absence of a touch screen is not good for the usability because the customer is used to have it

The embedded graphic engine will allow only simple animation

Project Target #1

The main goal to the project is to realize the new generation of hob user interfaces, allowing the user to have a better experience interacting with something that is more similar to a SMART device than a traditional user interface

And basically similar to other UI used for ovens

Project Target #2 (wishlist)

What is required from the market ?

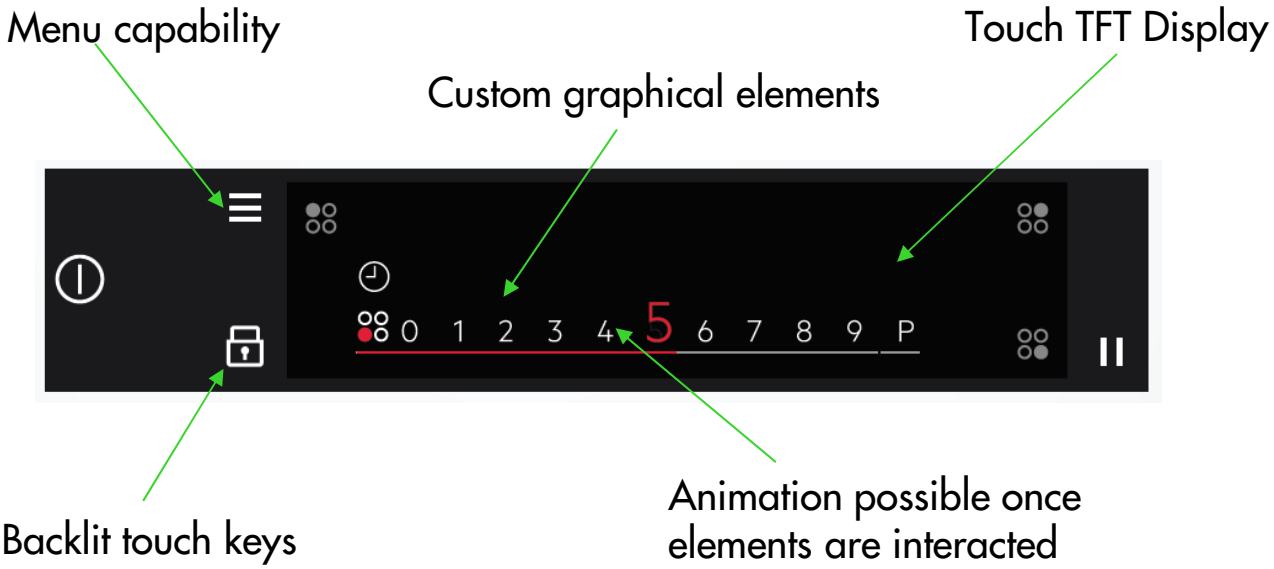
- Intuitive user experience
- Modern look and feel
- Assisted cooking in order to provide the best cooking performance
- Capable to support different type of induction hobs

What does it mean from technical point of view?

- TFT Display
- Full touch Display
- Graphical application
- Fancy animations
- Fluid interaction

Project Target #3 (Concept idea)

The previous wishlist led us into this concept: a microprocessor based user interface powerful enough to run a dedicated graphical library



Why don't use Qt?

Legacy SW architecture

The old software was deigned to be implemented into a microcontroller device.

This constrain imply a really simple SW architecture designed to use in the most efficient way all the (few) resources provided by the microcontroller that means:

- C as programming language
- Really simple SW main structure (simple scheduler or super loop)
- Minimize the memory footprint

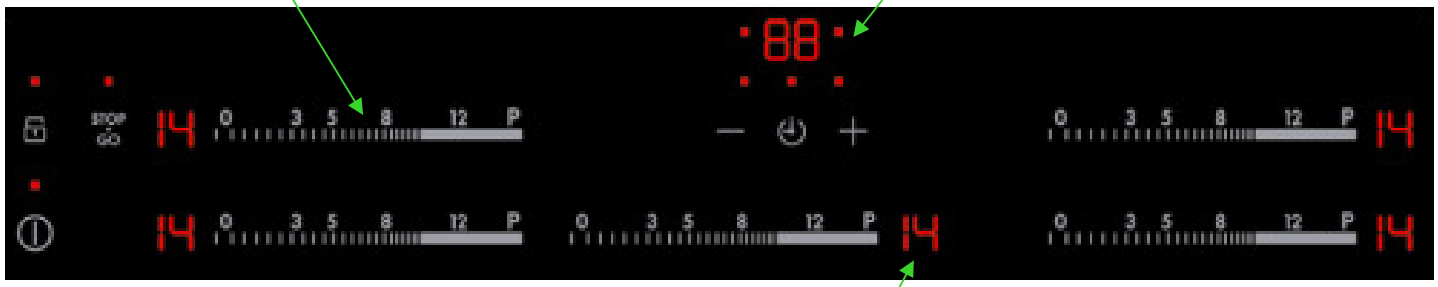
Except this physical constrains some of the main enemies of the SW engineering are:

- Strong HW dependency
- "Weak" modularization of the code
- Mixed SW responsibility in the same "module"

Legacy SW VS SW engineering?

Is the touch key management interfaced in a clean way with the main state machine ?

Is the LED management completely independent from the main state machine?



Are the shown values obtained by using getter?

New SW architecture C++ paradigms

Considering Qt is based on C++ this is a good chance to restructure the embedded legacy code following the C++ paradigms.

Some example of key points we tried to achieve are:

- **Encapsulation:** Create better defined / divided SW modules with a clear interface (avoid `extern`!!!)
- **Class oriented:** Better definition and management of the complex data types (massive use `struct` with a own constructor and own methods)
- **Inheritance:** Avoid duplication of code due to shared behavior among the modules (usage of `virtual` functions)

New SW architecture WARNINGS

During the restructuring of the code one of the critical part to take in account is for sure related to the overhead caused by this restructure.
For sure the HW chosen in order to implement this kind of project will be powerful but the risk to cause inefficiency (basically graphical delay) due to a huge overhead required has to be considered during the implementation

Some pitfall can be:

- Massive STACK usage
- HEAP usage cause memory leak (may be not used in a C based application)
- Massive memory used due to more complex software

New SW architecture Split logic / Graphic

One of the most famous hint from Robert C. Martin (Uncle Bob) about clean code is:

"… one function one responsibility…"

This brought us to find a way to separate completely:

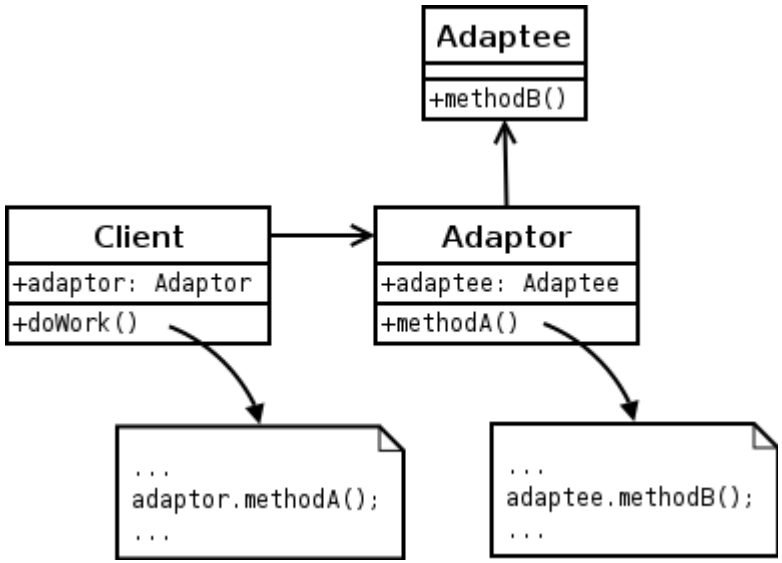
- logical part of the UI
- Graphical Part

The idea is to realize an interface to link the two different "worlds" with a common interface.

With this approach there's no need to share the knowledge of some functional details into the graphical parts and vice versa

How to split: Adapter

In software engineering, the adapter pattern is a software design pattern (also known as Wrapper, an alternative naming shared with the Decorator pattern) that allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.



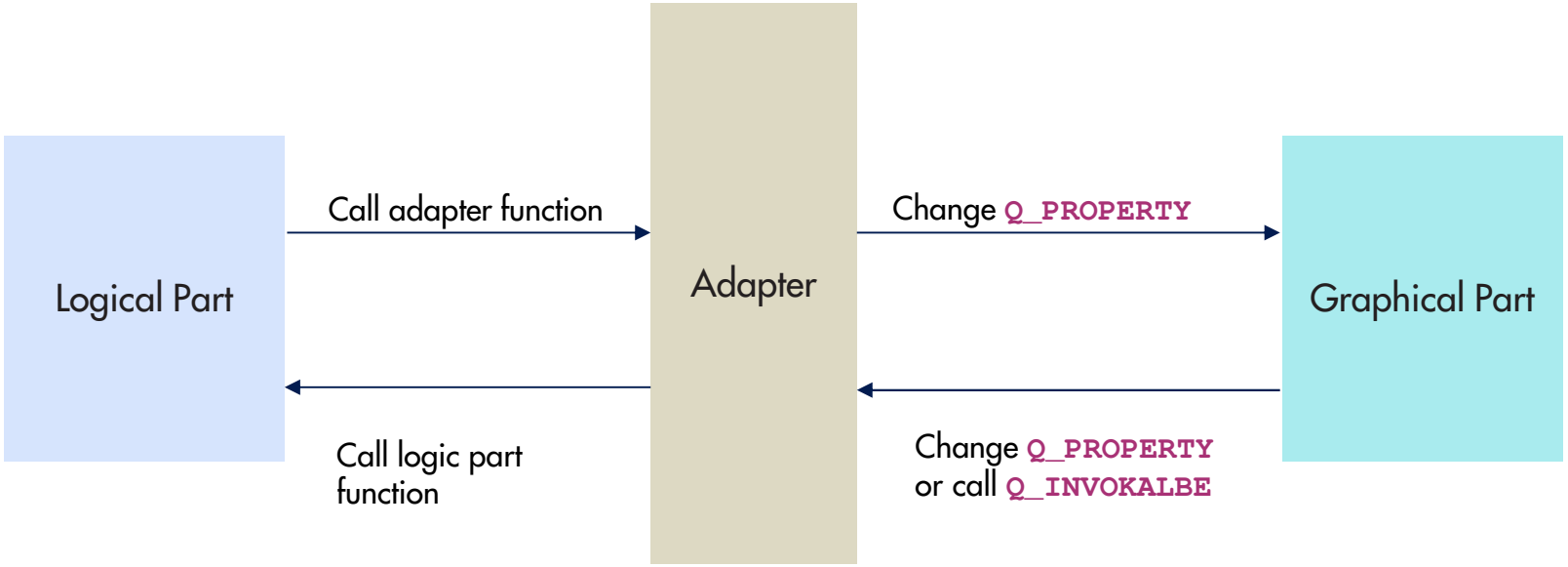
Source: Wikipedia

Adapter in Qt + QML

Considering Qt framework can provide a dedicated language for the graphical part (QML) this imply a really strong separation of the two world by using the **Adapter pattern**

Note: The adapter classes will developed by using features provided by Qt like:

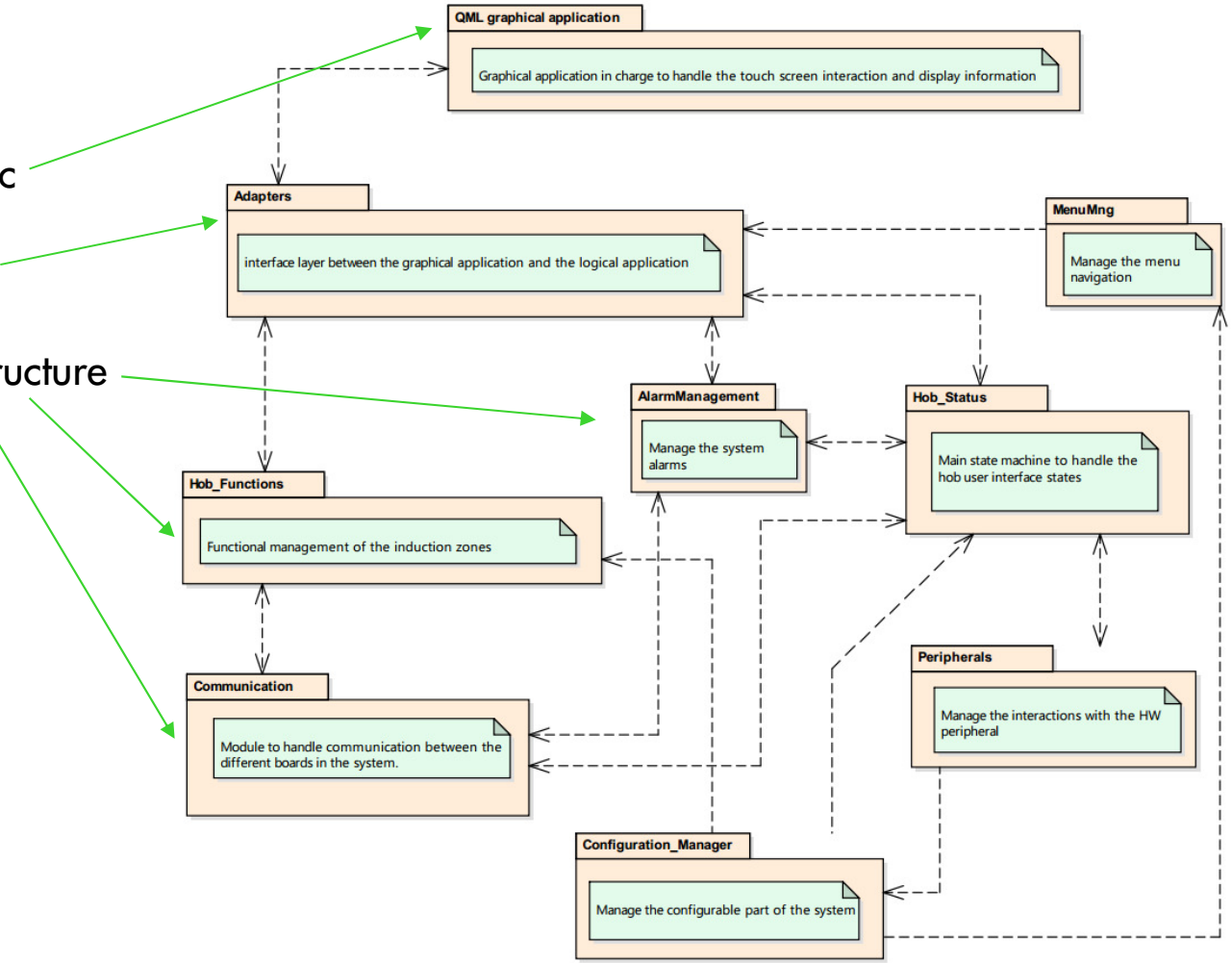
- `Q_PROPERTY`
- `Q_INVOKALBE`



Software architecture

Targets achieved:

- QML for graphic
- Adapter usage
- Modular SW structure

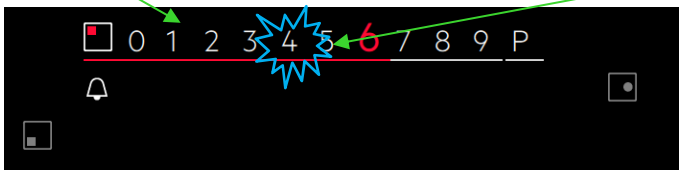


Application development #1

Interaction with a graphical element “click”

`Q_PROPERTY` or `Q_INVOKABLE` used to:

- inform the logical part
- Run animation (open slider)



New value selected “slide”

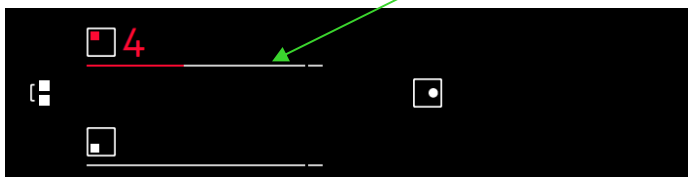
`Q_PROPERTY` or `Q_INVOKABLE` used to:

- inform the logical part
- Run animation (reduce red line)

Interaction with a graphical element “click” outside

`Q_PROPERTY` or `Q_INVOKABLE` used to:

- inform the logical part
- Run animation (reduce close slider)



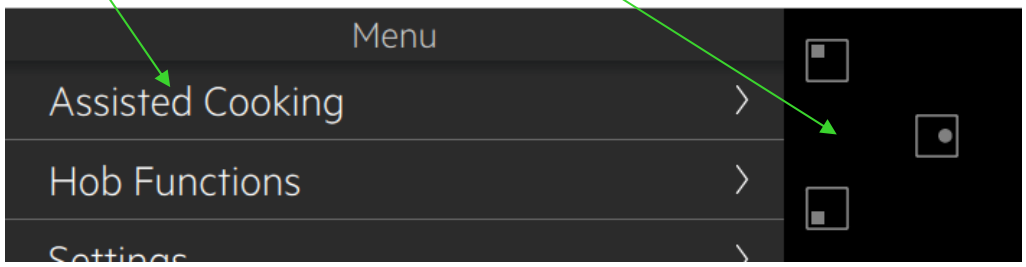
Application development #2

Interaction with an external touch key

Event generated by peripheral driver to inform the logic part

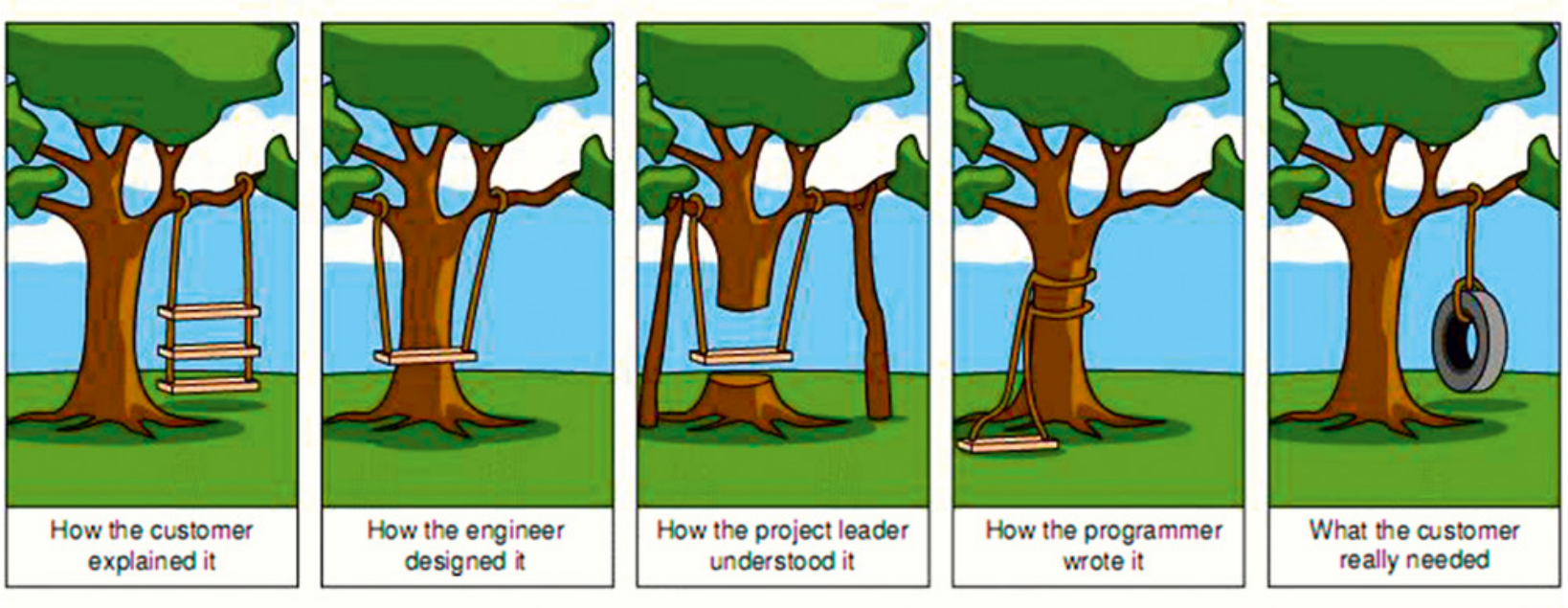
`Q_PROPERTY` or `Q_INVOKALBE` used to:

- Run animation (open menu squeeze icons)



Interaction with graphic designers

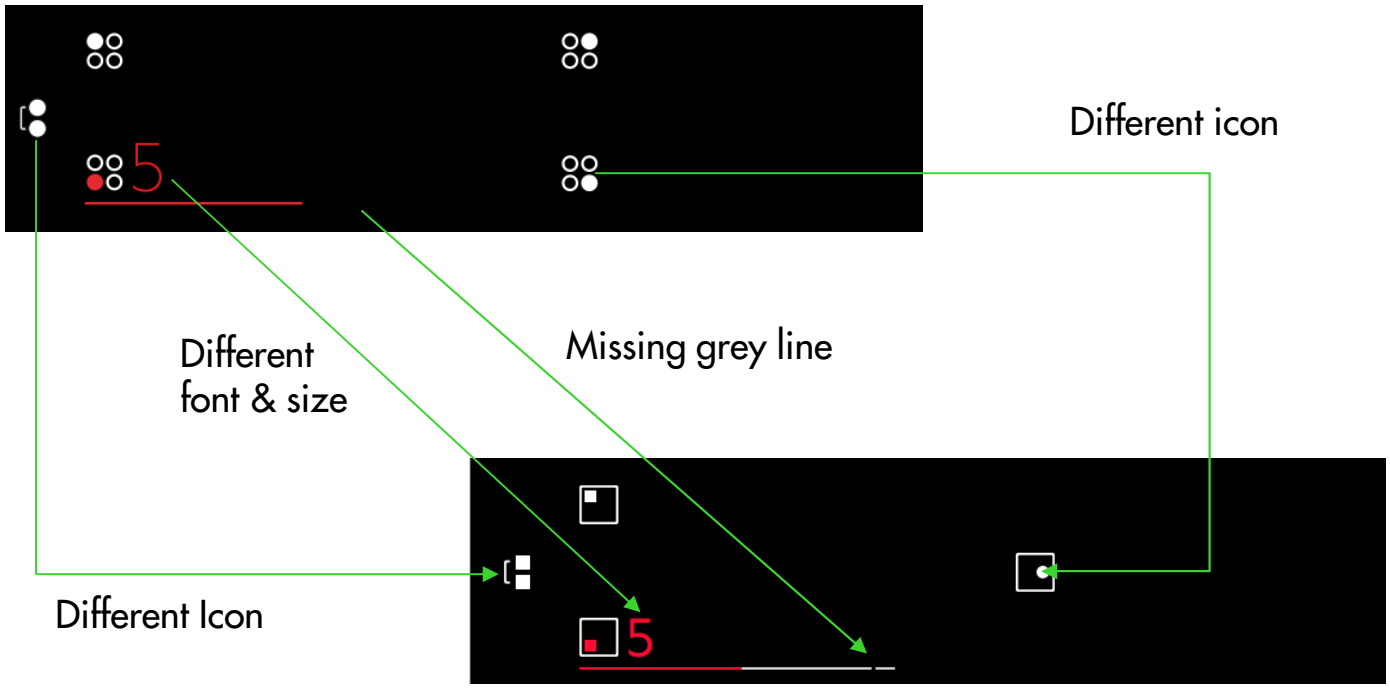
One of the most critical problem during the development of a graphical UI is related to the interaction with the graphical department, this can cause not nice misunderstanding



Source CMS garden

Practical example

Graphic specification Version X



Graphic specification Version X+1

How to allow safe graphic changes

The idea is to provide a way for the graphical department to:

- Modify a component in the easiest way (no QML coding)
- Avoid to involve (too much) the software developer
- Test the change independently
- Avoid the usage of special tool (e.g. in circuit programmer)

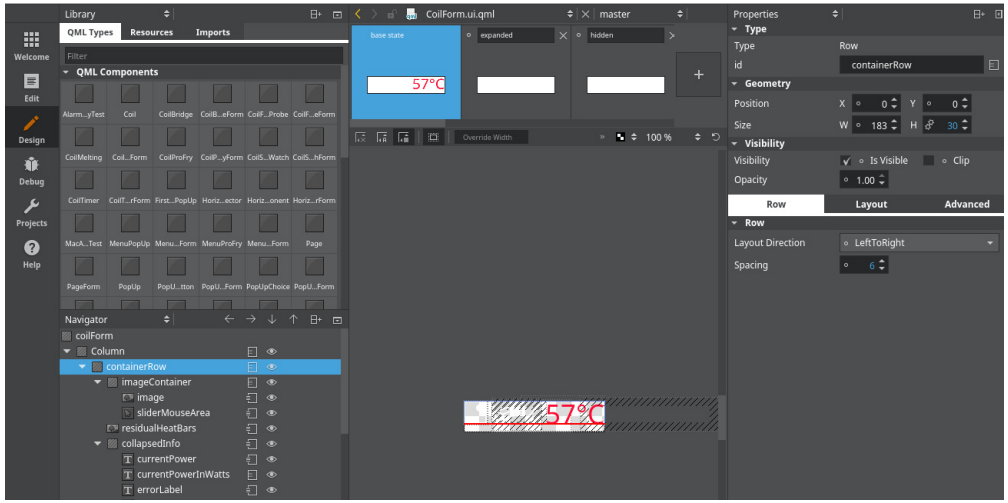
How to achieve it ?

1. Split the development in two QML file:
2. FileNameForm.ui.qml
3. FileName.qml will be a standard QML file

File FileNameForm.ui.qml editable only with QtDesigner

The file CoilForm.ui.qml will be the file editable only by the graphic design tool of Qt Creator this file will contain only graphical information (no scripting, no functions)

Qt Creator open this file using the designer by default this allow safe graphic change



If the edit mode is selected the IDE inform the user about possible risks

File FileName.qml
The logic behind the graphic

The file Coil.qml will be the file editable only by the text editor of Qt Creator this file will contain only the logical part relate to the component

Note CoilForm.ui.qml file

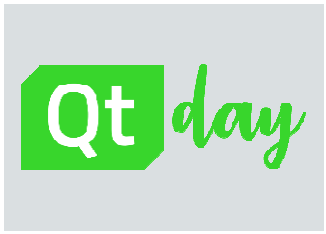
```
/* ============================================================ */
/*                        Import Folders                        */
/* ============================================================ */
 import QtQuick 2.4
 import hob.adapter 1.0

/* ============================================================ */
/*                        Main ITEM reference                   */
/* ============================================================*/
 CoilForm {
    id: coil
```

Developing tricks

1. Take advantage of the Qt multiplatform, very often the development is Linux based this feature allow other people to review the development steps in a windows machine, intercepting in an early stage some possible unwanted behaviors sparing time and effort for the dev team
2. Implement a PC simulator, even if can seems a extra task this allow to debug without the real board or the real appliance
3. Combine the point 2 with the .ui.qml files in order to improve the independency of the graphic department

The final result

After a lot of effort here the product



THANK YOU