Luca Ottaviano
Tech lead

Sviluppo di interfacce Qt su Raspberry

Develer workshop

20 NOVEMBRE 2019

develer

COS'È QT QUICK

Qt Quick è un sistema per sviluppare UI fluide

Permette di interagire con il sistema grazie all'integrazione con C++

È di proprietà di The Qt Company ed è rilasciato con doppia licenza open source e commerciale

PERCHÈ QT QUICK

È un framework “batterie incluse”

È uno standard di fatto in ambito Linux embedded industriale

È una soluzione cross-platform efficace anche su desktop e microcontrollori

L’obiettivo di stasera

Cruscotto

develer

PROGRAMMA

- Montaggio componenti e asset UI
- Integrazione con C++
- Compilazione su Raspberry e cenni alla cross compilazione
- Visualizzazione su browser

Insieme di tecnologie per lo sviluppo rapido di applicazioni

- QML: linguaggio dichiarativo per scrivere componenti UI
- Runtime: esegue il codice QML e fornisce il motore Javascript per eseguire il codice
- C++: backend per l'integrazione con la macchina

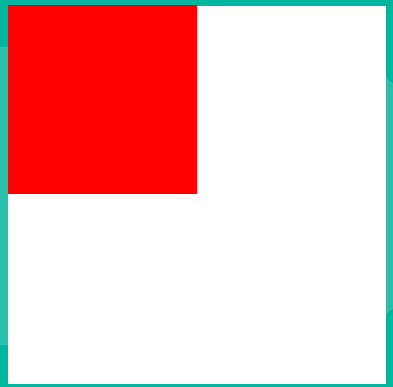
QML

- Descrive i componenti che fanno parte della UI
- Gestisce l'input utente (tramite mouse, touchscreen o tastiera)
- Esegue in una sandbox

C++

- Comunica con il QML tramite oggetti “promossi”
- Interagisce con la macchina
- Codice unrestricted

QML: CONCETTI DI BASE



```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```


develer

Blocco degli import

Istanziazione degli oggetti

```
import QtQuick 2.11
```

```
Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

Questo codice istanzia due oggetti di tipo Rectangle

I due oggetti sono in una scena

La posizione di istanziazione definisce anche la posizione in gerarchia

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

Attenzione, questi non sono comandi di disegno, ma istanziazioni

Un file QML viene interpretato per definire la scena

La scena viene poi disegnata quando serve

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

Ogni oggetto può avere al più un parent

L'oggetto senza parent è l'oggetto root

1 file QML = 1 oggetto root

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

Identificatore univoco oggetto QML

Inizia per lettera lowercase o _

È univoco all'interno del file QML

Serve agli oggetti per riferirsi tra loro

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

Ogni tipo QML espone delle property

Le property:

- hanno un tipo
- controllano aspetto e comportamento dell'oggetto

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

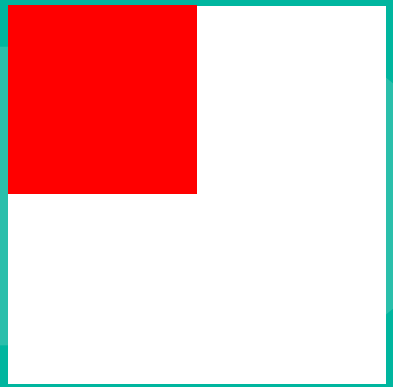
Le property possono essere in binding tra loro

Il binding descrive la relazione tra la property e un'espressione

Ogni volta che l'espressione a destra cambia, la property viene rivalutata

```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```

QML: CONCETTI DI BASE



```
import QtQuick 2.11

Rectangle {
    id: root
    color: "white"
    width: 200
    height: 200
    Rectangle {
        color: "red"
        width: root.width / 2
        height: root.height / 2
    }
}
```


develer

TIPI DELLE PROPERTY

Forniti da Javascript

- int, bool, real, double
- string, url, list, var

Forniti da Qt Quick

- color, font, date, time, point, size, rect
- matrix4x4, vector2d, vector3d

TIPI DI OGGETTI QML

Tipi visuali

- Item (ha una superficie ma non un aspetto)
- Rectangle, Text, Image
- Row, Column (non hanno aspetto proprio)

Tipi di input

- MouseArea

Per creare maschere complesse si assemblano insieme oggetti di tipo più semplice

Ci sono tre modi principali di posizionare oggetti a video:

- Contenitori (oggetti Row, Column)
- Ancoraggi
- …anche posizionamento assoluto (x, y)

CONTENITORI


```
import QtQuick 2.11

Row {
    spacing: 10
    Image {
        source: "images/D.png"
    }
    Image {
        source: "images/N.png”
    }
    Image {
        source: "images/P.png"
    }
}
```


develer

CONTENITORI

Controllano posizione degli elementi figli

Nessuna autorità su dimensioni

I contenitori sono oggetti “logici”, non hanno aspetto visuale

```
import QtQuick 2.11

Row {
    spacing: 10
    Image {
        source: "images/D.png"
    }
    Image {
        source: "images/N.png”
    }
    Image {
        source: "images/P.png"
    }
}
```

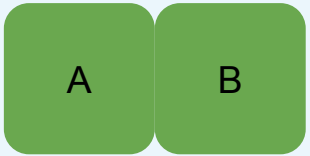
ANCORAGGI

Sono il secondo metodo di posizionamento

Controllano la posizione e la dimensione degli elementi

Sono basati sul concetto di **property binding**

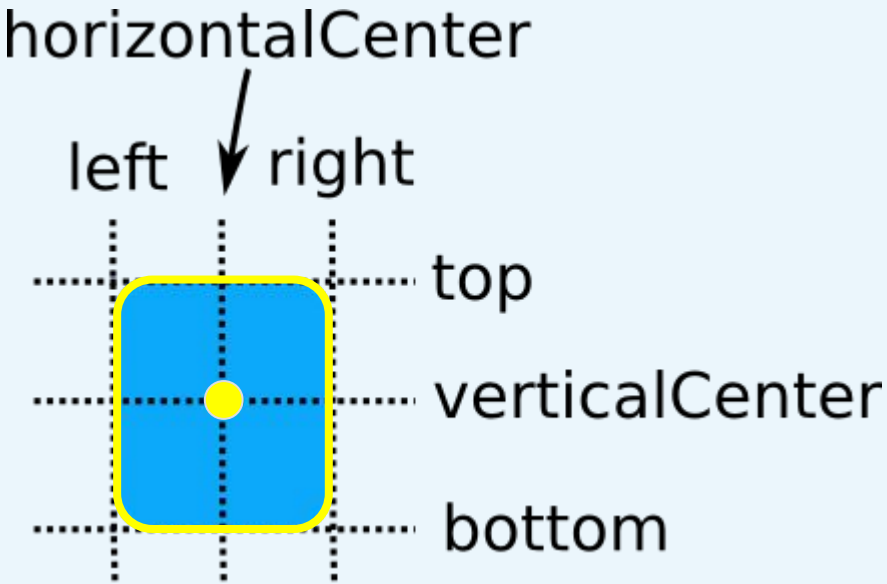
ANCORAGGI



```
Item {
    id: b
    anchors.left: a.right
}
```

```
Item {
    id: small
    anchors.centerIn: big
}
```

ANCORAGGI



Margini: topMargin, leftMargin

fill e centerIn: scorciatoie

ANCORAGGI

0 %

```
import QtQuick 2.11

Item {
    id: batt
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: "0 %"
    }
}
```

ANCORAGGI

Le linee di ancoraggio vanno in binding con le linee di ancoraggio.

Linee orizzontali vanno con linee orizzontali; linee verticali vanno con linee verticali

```
import QtQuick 2.11

Item {
    id: batt
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: "0 %"
    }
}
```


develer

ANCORAGGI

I margini sono attivi solo se è definita la rispettiva linea di ancoraggio

```
import QtQuick 2.11

Item {
    id: batt
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: "0 %"
    }
}
```

COMPONENTI

I componenti servono per comporre insieme oggetti più semplici

Sono il modo principale con cui si riusa il codice in QML

Vediamo come possiamo rendere riusabile il componente batteria

```
import QtQuick 2.11

Item {
    id: batt
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: "0 %"
    }
}
```

Aggiungiamo una property all'elemento root

Le property dell'oggetto root fanno da interfaccia per il componente

Salviamo il file con iniziale maiuscola (Battery.qml)

```
import QtQuick 2.11

Item {
    id: batt

    property int battery: 0
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: batt.battery + "%"
    }
}
```

COMPONENTI

```
Item {
    id: mainWindow
    Battery {
        battery: control_unit.battery
    }
}
```

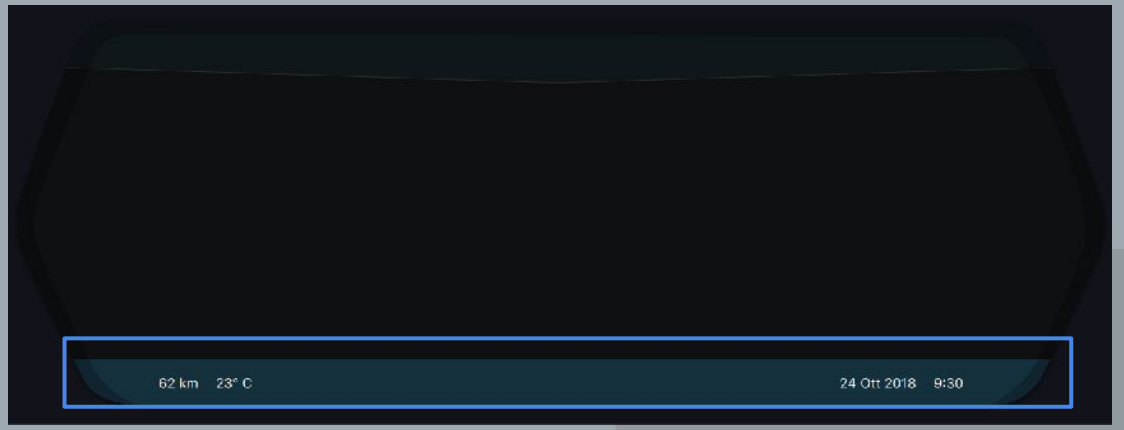
main.qml

```
import QtQuick 2.11

Item {
    id: batt

    property int battery: 0
    width: 200
    height: 200
    Image {
        id: batt_img
        anchors.top: batt.top
        anchors.horizontalCenter: batt.horizontalCenter
        source: "assets/images/batteria_vuota.png"
    }
    Text {
        anchors.top: batt_img.bottom
        anchors.topMargin: 10
        anchors.horizontalCenter: batt.horizontalCenter
        text: batt.battery + "%"
    }
}
```

Battery.qml

CRUSCOTTO

Realizziamo insieme il primo componente



develer

ESERCIZIO

Realizzate la ghiera della velocità, usando queste immagini:

- contagiri_back
- contagiri_top
- ghiera+numeri
- centrale

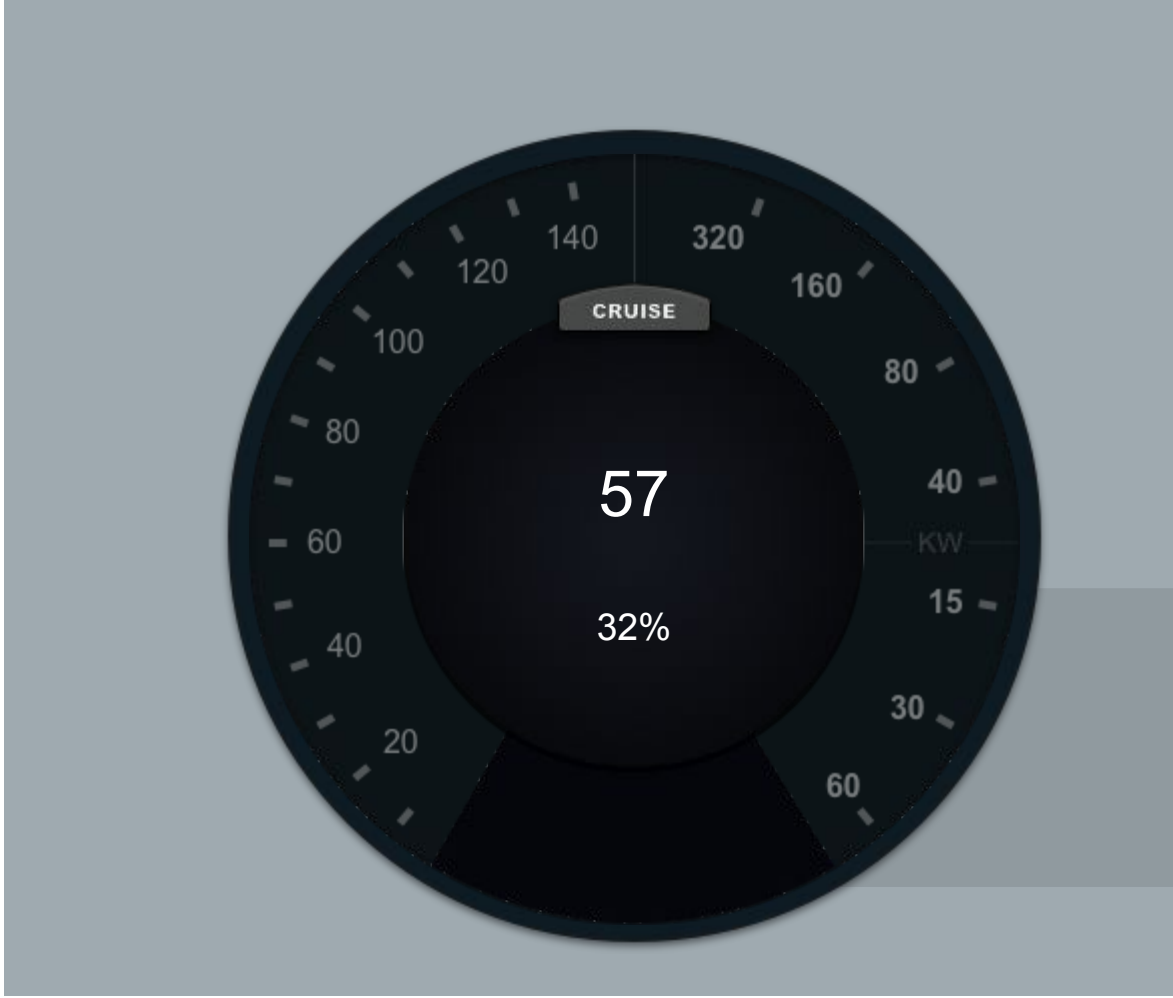
develer

ESERCIZIO

Inserite la velocità e il livello di batteria dentro la ghiera usando dei componenti Text:

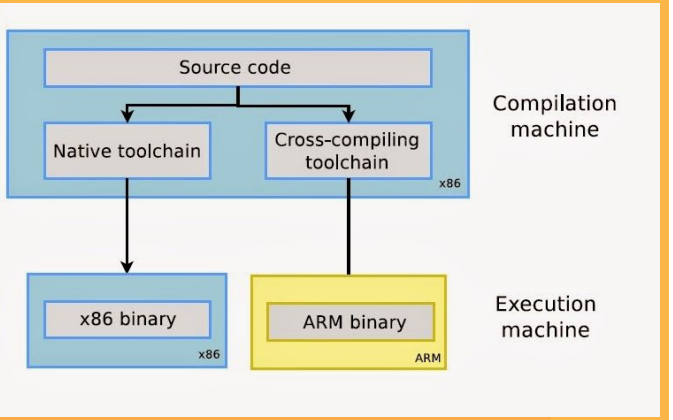
- Velocità (speed)
- Livello di batteria (battery)

Entrambi sono esposti come property della control_unit



develer

CROSS COMPILAZIONE



Con cross compilazione si intende l'uso di un compilatore su PC (HOST) che generi codice per la scheda embedded (TARGET)

Si usa perché:

- La macchina HOST è molto più potente (in genere)
- Non vogliamo avere il compilatore su TARGET
- Non vogliamo avere gli header su TARGET

Non la useremo perché:

- Il TARGET Raspberry richiede un HOST Linux
- Il codice è poco, si può compilare su TARGET

COMPILAZIONE SU RASPBERRY

Copiamo i file su Raspberry
e poi compiliamo
l'applicazione

```
host $ rsync -avz workshop-qt-webgl
pi@raspberrypi.local:/home/pi --exclude .git
host $ ssh pi@raspberrypi.local

target $ cd /home/pi/workshop-qt-webgl
target $ qmake
target $ make
target $ ./workshop
```

ESECUZIONE DEL PROGRAMMA

Per visualizzare il cruscotto sul display

```
target $ export DISPLAY=:0
target $ ./program-name
```

COSA È IL PLUGIN WEBGL STREAMING

Il plugin Qt Quick WebGL è un platform plugin per accedere a interfacce Qt Quick via rete

La UI è renderizzata sul browser tramite WebGL

L'accesso è single user, in caso di più connessioni contemporanee solo il primo utente può accedere

develer

COME CONNETTERSI

Lanciare il programma con il parametro giusto target $ ./cruise -platform webgl

Collegarsi tramite browser da host: http://raspberrypi.local:8080

CONTACTS

Luca Ottaviano

luca.ottaviano@develer.com
Twitter: @lucaotta

www.develer.com