Université de la Rochelle
Licence d'Informatique - IHM (INFO132601)

TP 5 - *Syntaxe déclarative des IHM: Qt Quick et QML*

© B. Besserer, R. Péteri

Année universitaire 2016-2017

En plus des habituels points de validation au cours du TP, vous ferez un rapport comprenant les parties de codes à rajouter, ainsi que des captures d'écran de l'IHM. Ce rapport (1 par binôme) est à remettre sous Moodle pour le 8 mars dans un seul fichier pdf .

Préambule

Ce TP porte sur une manière déclarative de concevoir facilement des IHMs avec QtQuick et QML. N'ayant pas eu le temps de le traiter dans le cadre de l'UE, nous vous indiquerons des tutoriels, et ce TP permettra de mettre en pratique ce que vous aurez vu.
On pourra donc se référer pour commencer une introduction à QML au tutorial : `http://qt.developpez.com/doc/4.7/qml-tutorial/` notamment le lien pour définir de nouveau composants : `http://qt.developpez.com/doc/4.7/qml-extending-types/#defining-new-components`.
Enfin, vous regarderez ce tutorial portant sur l'intégration avec Qt Quick de QML dans du code C++ : `http://qt-devnet.developpez.com/tutoriels/qt-quick/pour-developpeurs-cpp/`.
N'hésitez pas à regarder d'autres tutoriaux suivant vos besoins.
Une fois pris en main QML et Qt Quick, vous allez appliquer vos connaissances sur deux exercices.

1 QML et Qt Quick

1.1 Préparation

Le TP est fait pour être réalisé avec **Qt5**. Créer un nouveau projet de type `Qt Quick Controls Application` avec Qt Creator (si aucun Kit n'est trouvé, cliquez sur options et vérifié que la version de Qt utilisée est bien Qt5 et non Qt4).
Observer l'organisation des fichiers et leurs contenus créé par Qt Creator. Prenez le temps de bien comprendre les différents imports ainsi que le lien entre les fichiers `main.cpp` (appel de l'IHM) et `main.qml` (description de l'IHM avec le language QML). La fenêtre prédéfinie est de type `ApplicationWindow` (équivalent de `QmainWindow` avec barre de menu et statut), les éléments sont dessinés les uns après les autres (attention aux éléments écrasés ! ! !).

1.2 Initiation au langage QML

1.2.1 Identifier les composants

Executer tout d'abord le programme pour vérifier que l'interface par défaut (un menu et deux boutons cliquables) est bien crée.
Dans le code du fichier `main.qml` effacer les lignes :

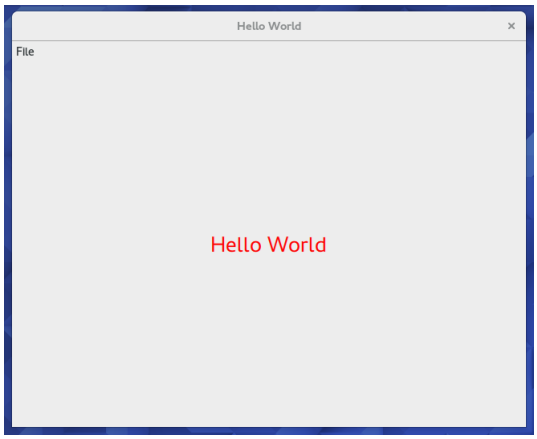
```
button1.onClicked: messageDialog.show(qsTr("Button 1 pressed"))
button2.onClicked: messageDialog.show(qsTr("Button 2 pressed"))
```

associant aux 2 boutons leurs actions, puis aller ensuite dans l'onglet `Design` pour effacer ces boutons.

Retourner dans le fichier `main.qml` puis remplacer au même endroit par la création d'un composant `Text` dans la fenêtre principal en mettant le code ci-dessous :

```
Text{
   text: qsTr("Hello World")
   color: "red"
   font.pointSize: 20
   anchors.horizontalCenter: parent.horizontalCenter
   anchors.verticalCenter: parent.verticalCenter
}
```

Vous devriez désormais avoir ceci :



Remplacer le texte « Hello World » par le texte « `Open clicked` » *lors d'un clic sur* `Open` *du menu* `File`. Aide : ajouter un identifiant au composant Text pour pouvoir y accéder dans la propriété `onTriggered` du composant `MenuItem` correspondant (`onTriggered: identifiant.text = "MonTexte";`).

1.2.2 Dessiner des formes

Remplacer le texte « Hello world » par deux rectangles de taille 200×300 pixels et espacés de 10 pixels, définis comme suit :

```
Rectangle{
        id : monRectangle
        width: 200
        height: ...
        anchors.left: ...
        anchors.leftMargin: ...
    }
```

1.2.3 Insertion d'images

Remplacer la couleur de fond des rectangles par les images `img1.jpg` et `img2.jpg`.
Conseils : ajouter les fichiers images au projet `Qt Quick` de manière à ce qu'ils soient accessibles par l'application, placer un composant Image à l'intérieur de chaque composant `Rectangle` et utiliser `anchors.fill: parent` pour que l'image remplisse la surface du composant parent.

```
    Rectangle{
        ...
        Image{
            id: ...
            anchors.fill: parent
            source:"monImage.jpg"
        }
    }
```

1.2.4 Insertion de texte

Ajouter le texte « Image 1 » et « Image 2 » centré en haut du rectangle 1 et 2 respectivement. Le texte sera en taille 20 et de couleur blanche.
Aide :
insérer un composant `Text` dans chaque composant `Rectangle` et utiliser les propriétés :
`anchors.horizontalCenter: parent.horizontalCenter` et `font.pointSize`. La couleur peut être définit par son code hexadécimal.

1.2.5 Insertion de boutons

Insérer un bouton au centre de chaque rectangle contenant le texte "Changer".

1.2.6 Interaction avec la souris

On souhaite diviser la transparence des rectangles par 2 lors de survol de la souris. Pour cela, utiliser le composant `MouseArea` à l'intérieur des composants `Rectangle` (on regardera l'aide des signaux `entered()` et `exited()`).


1

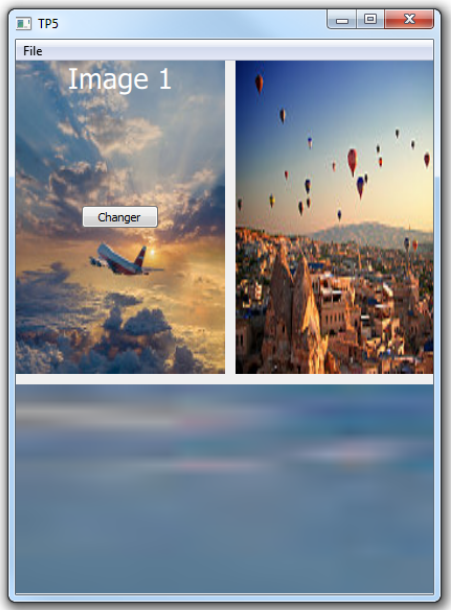
1.2.7 Boîte de dialogue

Créer un composant `FileDialog` (boîte de dialogue) pour permettre à l'utilisateur de sélectionner un fichier de type image sur le disque lors d'un clic sur les boutons "Changer".
Aide : ajouter la propriété `onClicked : maBoiteDeDialogue.open()` aux boutons pour lancer l'action sur événement du clic. On pourra regarder l'aide en ligne : `http://doc.qt.io/qt-5/qml-qtquick-dialogs-filedialog.html`

1.2.8 Mise à jour de propriétés

Après sélection de la nouvelle image via la boîte de dialogue, elle doit venir remplacer l'image de fond du rectangle concerné.
Aide : utiliser la propriété `onAccepted` du composant `FileDialog` pour changer la source du composant `Image` correspondant (`onAccepted: img1.source = fileUrl`)

1.2.9 Déformation d'images



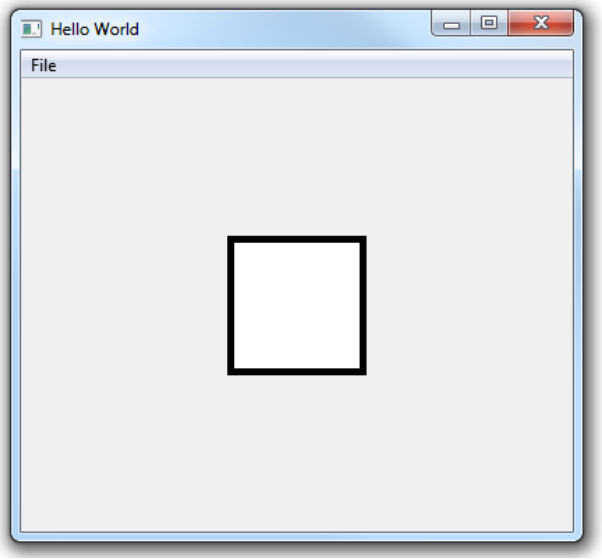
Les composants peuvent facilement devenir réactif au mouvement de la souris (ou doigt pour les interfaces tactiles) lorsqu'ils sont insérés dans un composant de type `Flickable`. On souhaite unsérer une nouvelle image au-dessous des deux autres de manière à faire en sorte qu'elle soit plus grande que la zone du composant Flickable, et que l'on puisse ainsi *scroller* dedans. En suivant l'exemple ci-dessous, essayer de déplacer l'image en cliquant dessus avec la souris ou avec le pad.

```
Flickable{
    id:flickImg2
    anchors.top:...
    width:...
    height: ...
    contentWidth: img3.width
    contentHeight: img3.height
    clip:true
    Image{
        id:img3
        source:"img3.jpg"
     }
}
```

2

2 QML et états

Dans un nouveau projet de type Qt Quick Controls Application, créez un rectangle de taille 100x100 (*width*, *height*), avec un fond blanc (*color*), des bordures noires d'épaisseur 5 (*border.color*, *border.width*), et qui sera centré à l'intérieur de votre fenêtre (*anchors*). Voici ce que vous devriez voir :



Nous allons utiliser ce rectangle comme base pour définir des états QML et modifier son apparence en réponse à une interaction de l'utilisateur.

2.1 Les états

Un état est une abstraction des propriétés d'une collection d'éléments. Les interfaces utilisateurs sont conçues pour présenter différentes configurations d'interface dans différents scénarios ou pour modifier leur apparence en réponse à une interaction utilisateur. Souvent, un ensemble de changements est effectué simultanément, tel que l'interface puisse être vue comme changeant en interne d'un état à un autre.
Une des caractéristiques des états QML est la capacité d'un élément à retourner à son état par défaut.
On pourra se référer à : http://qt.developpez.com/doc/4.7/qml-tutorial3/.

In QML, states are a set of property configurations defined in a State element. Different configurations could, for example :

- *Show some UI elements and hide others*
- *Present different available actions to the user*
- *Start, stop, or pause animations*
- *Execute some script required in the new state*
- *Change a property value for a particular item*
- *Show a different view or screen*

All Item-based objects have a state property, and can specify additional states by adding new State objects to the item's states property. Each state within a component has a unique name, an empty string being the default. To change the current state of an item, set the state property to the name of the state.

2.2

Le code suivant permet la gestion d'un état « sélection » en réponse au clic gauche souris.

```
import QtQuick 1.0

Rectangle {
    id: afficheur

    border.color: "black"

    color: "white"
    border.width: 5
    width: 100
    height: 100

    MouseArea {
        anchors.fill: parent

        acceptedButtons: Qt.LeftButton
        onClicked: {
            afficheur.state = 'selection';
        }
    }

    states: [
        State {
            name: "selection"
            PropertyChanges {
                target: afficheur
                        ..........................
            }
        }
    ]
}
```

— Ajouter le code permettant la modification de la propriété par défaut de la couleur de la bordure du rectangle dont l'identifiant est « afficheur ».

— Modifier le code dans le composant `MouseArea` de façon à associer un état 'sélection' au clic gauche souris, et l'état par défaut sur un clic droit.
Aide : pour retourner à l'état par défaut : `id_composant.state=' ';`
La propriété `mouse.button` permet d'accéder au bouton associé au clic souris.


3

2.3

On veut maintenant agrandir le rectangle lorsqu'il est survolé par la souris. Le survol correspond à l'action « `onEntered:` » et « `onExited :` » de `MouseArea`, et doit être activé grâce à la propriété : « `hoverEnabled: true` ».

— Modifiez le code de façon à ajouter la gestion d'un nouvel état « agrandissement », sur l'action « `onEntered` » de `MouseArea`, et le retour à l'état initial sur l'action « `onExited :` » de `MouseArea`. On pourra par exemple pour tester modifier la taille du rectangle.

2.4

Les changements d'états peuvent être animés à travers des **transitions**.
Le code suivant permet l'ajout d'une animation lors de la modification des attributs x ou y d'un élément.

```
transitions: [
         Transition {
             NumberAnimation { properties: "x,y"; duration: 500 }
         }
     ]
```

Ajoutez une animation lors de l'agrandissement du rectangle (on veut doubler la hauteur et la largeur du rectangle).

2.5

Par défaut, les propriétés non modifiées dans l'état ont comme valeur, la valeur par défaut du Rectangle. Il pourrait être intéressant notamment pour la sélection de conserver le changement de couleur lors du clic gauche de la souris.

Définir une propriété de type booléen permettant de stocker les modifications de la sélection.

 4 Modifier dans chaque état les changements de propriétés couleur et taille en conséquence de façon à conserver la sélection sauf sur un clic droit qui appelera toujours l'état par défaut.