MCM440-QML

Application Structure:

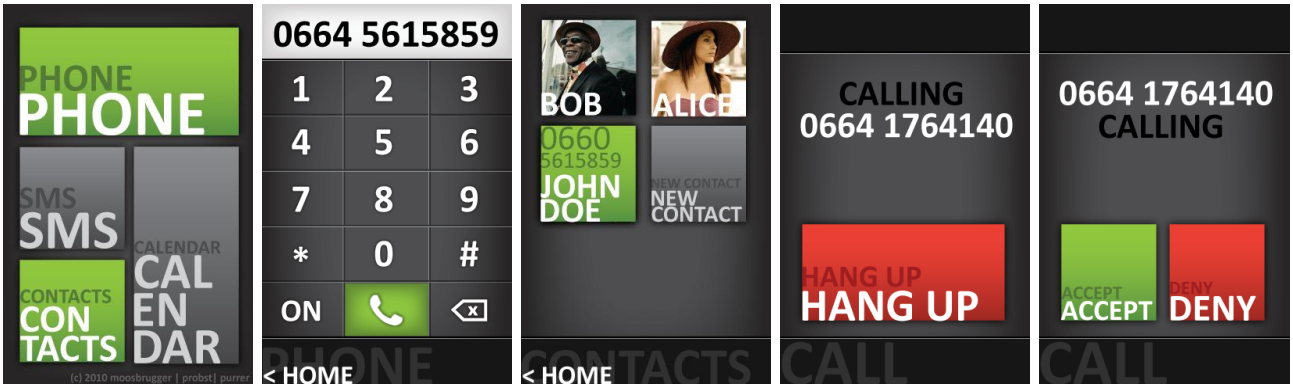

Figure 1: mcm440-phone-app components - home HomeComponent, DialerComponent, AddressbookComponent, CallHandlingComponent (outgoing/incoming call)

Our MCM440 application consists of four components (see figure 1):

- **HomeComponent**
  A home screen containing several app icons. Elements colored in green (Phone, Contacts) are active elements and launch the matching component upon selection. Elements colored in gray are dummy elements without any functionality.
- **DialerComponent**
  A dialer screen for dialing and launching phonecalls.
- **AddressBookComponent**
  An address book screen containing three contacts. Upon selection of a contact's photo, the matching phone number is shown. Selecting the contact again, launches a call.
- **CallHandlingComponent**
  A call screen for outgoing/incoming calls with functionality for accepting/denying resp. ending calls.

These four components are combined in **PhoneAppMain.qml**, which is also responsible for shifts between the different components. Therefore, the main app manages four states ("homeState", "dialerState", "addressBookState", "callState"), which manipulate the alpha values of the components. Corresponding to that, transitions (example see figure 2) were defined, which add smoothly animate the alpha changes.

```
states: [
    State {
        name: "homeState";
        when: phoneAppMain.showComponent == "home";
        PropertyChanges {
            target: home; opacity : 1
        }
        PropertyChanges {
            target: dialer; opacity : 0
        }
        PropertyChanges {
            target: addressBook; opacity : 0
        }
        PropertyChanges {
            target: call; opacity: 0
        }
    },
```

```
transitions: [
    Transition {
        to: "homeState"; reversible: true
        NumberAnimation { properties: "opacity"; duration: 500 }
    },
```

Figure 2: PhoneAppMain.qml States example (left), PhoneAppMain.qml Transitions example (right)

QML-Resources:

HomeComponent:

The home screen consists of four app icons, of which two are active elements launching the DialerComponent resp. AddressBookComponent.

Main QML resources for the HomeComponent:

- **HomeComponent.qml**
  The main part of the HomeComponent with a resolution of 320x480 pixels, which determines the layout and contains several *HomeButton* elements. This file also specifies the actions to be invoked in response to *onClicked* signals of the home buttons.
- **HomeButton.qml**
  An app icon consisting of an *Image*- and a *MouseArea*-element. Properties: backgroundimage (string), enabled (bool)

DialerComponent:

The dialer screen consists of a simple number pad, a display showing the phone number entered by the user as well as a power-, a call- and a clear-button.

Main QML resources for the DialerComponent:

- **DialerComponent.qml**
  The main part of the DialerComponent with a resolution of 320x480 pixels, which determines the layout and contains several *DialerKey* elements as well as a *DialerDisplay* element. This file also specifies the actions to be invoked in response to *onClicked* signals of the dialer keys.
- **DialerDisplay.qml**
  A display in the top area of the screen consisting of an Image- and a Text-element.
  Properties: text (string)
- **DialerKey.qml**
  A single digit of the number pad, consisting of an *Image*-, a *Text*- and a *MouseArea*-element.
  Properties: text (string), fontsize (int), backgroundimage (string), icon (string)

AddressBookComponent:

The address book screen consists of three contact entries as well as a (dummy) button for the creation of new contacts.

Main QML resources for the AddressBookComponent:

- **AddressBookComponent.qml**
  The main part of the AddressBookComponent with a resolution of 320x480 pixels, which determines the layout and contains several *AddressEntry* elements. This file also specifies the actions to be invoked in response to *onClicked* signals of the address entries.
- **AddressEntry.qml**
  A single contact address consisting of two *Image*- and one *MouseArea*-elements. Upon selection, the element can transition between two states (front = contact photo, back = phone number). Properties: image_front (string), image_back (string), currentImage (string)

CallHandlingComponent:

The call screen consists of a text area displaying the number of the called person as well as controls for ending (active call) resp. accepting/denying (incoming call).

Main QML resources for the CallHandlingComponent:

- **CallHandlingComponent.qml**
  The main part of the CallHandlingComponent with a resolution of 320x480 pixels, which determines the layout and contains several text- and button-elements (HomeButton.qml reused). This file also specifies the actions to be invoked in response to the *onClicked* signals of the buttons. In addition, the component specifies three different states for outgoing ("outgoingState") and incoming ("incomingState", "incomingActiveState") to be changed via the call_type-property. Properties: call_type (string), number (string)

Combination of QML and C++

In order to be able to invoke C++ methods in QML files, the following steps were accomplished:

1) Adding *Declarative* module in the project file mcm400-phone-app.pro:

```
QT += declarative
```

2) Using *DeclarativeEngine*, *DeclarativeContext* and *DeclarativeView* in the main method:

```
QApplication a(argc, argv);
QDeclarativeView view;
view.rootContext()->setContextProperty("OfonoContext", new gsm());
view.setSource(QUrl("qrc:/qml/PhoneAppMain.qml"));
view.show();
```

The setContextProperty() method determines, which objects to be provided in QML. In our case, this was an *gsm* object called "OfonoContext". Another example could be:

```
setContextProperty("backgroundColor", QColor::black).
```

3) Signal/Slot definition in C++:

In order to be invokable in QML, slots have to be declared either have to be declared in the public slots section or marked using the Q_INVOKABLE macro. Signals are defined in the signals area (examples see figure 3).

```
class Gsm : public QObject
{
    Q_OBJECT

private:
    bool isConnected;

public:
    explicit Gsm(QObject *parent = 0);
    Q_INVOKABLE bool getModemStatus();

signals:
    void powerOn();
    void powerOff();
    void incomingCall(QString id);
    void outgoingCall(QString id);
    void endCall(QString id);

public slots:
    bool powerModemOn();
    bool powerModemOff();
    bool dialNumber(QString number);
    bool hangupAll();
    void modemPropertyChanged(const QString &name, const QDBusVariant &value);
    void voicecallManagerPropertyChanged(const QString &name, const QDBusVariant &value);
};
```

Figure 3: Gsm class definition

3) Method invocation in QML:

Slots can be invoked in QML using the previously defined context property (see figure 4), in our case „OfonoContext“ eg `OfonoContext.powerModemOn()`.

Signals can be caught using the *Connections* element, specifying the “OfonoContext” as target and specifying the actions to be taken for the specified signals (eg powerOn -> onPowerOn).

```
//power toggle button
DialerKey {
    id: key_power
    x: col1origin
    y: row5origin
    text: "ON"
    fontsize: 30
    Connections {
        target:OfonoContext
        onPowerOn: {
            key_power.text = "OFF";
        }
        onPowerOff: {
            key_power.text = "ON";
        }
    }
    onClicked: {
        if(key_power.text == "OFF") {
            OfonoContext.powerModemOff();
        } else {
           OfonoContext.powerModemOn();
        }
    }
}
```

Figure 4: power button in DialerComponent.qml