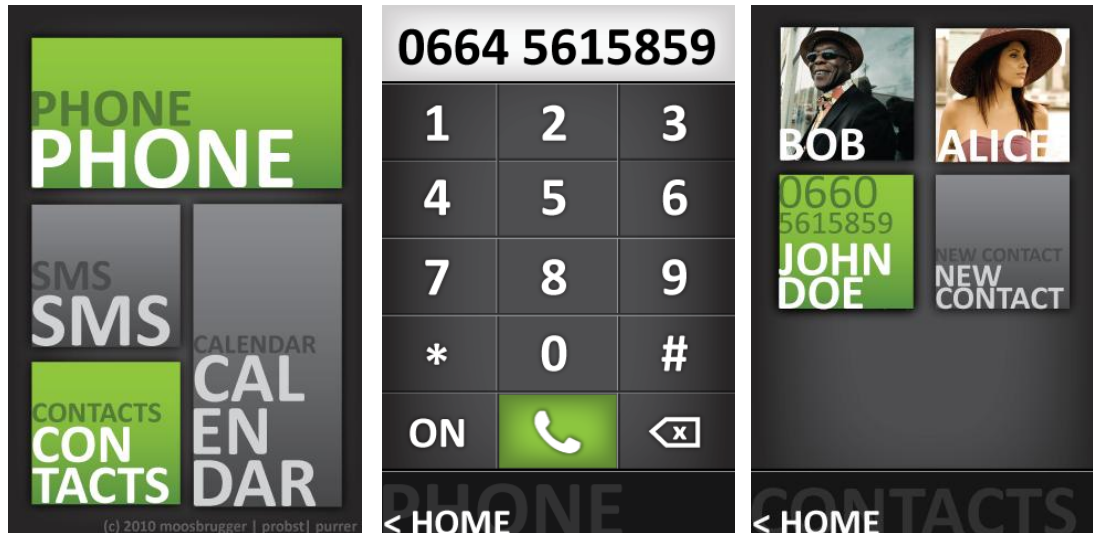


MCM440-QML

Application Structure:



Die Applikation besteht aus drei Komponenten:

- ▶ **HomeComponent**
Ein Home-Screen, der verschiedene App-Icons enthält. Grün eingefärbte Elemente (Phone, Contacts) sind aktiv und starten die jeweilige Funktion, grau eingefärbte Elemente sind Dummy-Elemente ohne Funktionalität.
- ▶ **DialerComponent**
Ein Dialer-Screen zum Absetzen und Empfangen von Anrufen.
- ▶ **AddressBookComponent**
Ein Adressbuch-Screen, der drei Kontakte enthält. Bei Selektion eines Kontaktfotos wird die dazugehörige Telefonnummer angezeigt, eine wiederholte Selektion startet einen Anruf.

Zusammengefasst sind diese Komponenten in der Datei **PhoneAppMain.qml**, die außerdem für das Umschalten zwischen den Komponenten zuständig ist. Realisiert wurde das in Form von drei **States** („homeState“, „dialerState“, „addressBookState“), die die Alpha-Werte der Komponenten manipulieren. Passend dazu wurden **Transitions** definiert, die für die Animation der Übergänge sorgen.

```
states: [
    State {
        name: "homeState";
        when: phoneAppMain.showComponent == "home";
        PropertyChanges {
            target: home; x : 0
        }
        PropertyChanges {
            target: dialer; x : 320
        }
        PropertyChanges {
            target: addressBook; x : 320
        }
    },
    transitions: [
        Transition {
            to: "homeState"; reversible: true
            NumberAnimation { properties: "x"; duration: 500 }
        },
    ],
]
```

HomeComponent QML-Ressources:

Der Home-Screen besteht aus vier App-Icons, von denen zwei aktiviert sind und jeweils eine der beiden anderen Komponenten (DialerComponent, AddressBookComponent) starten.

Umgesetzt wurde der Home-Screen in zwei QML-Dateien:

- ▶ **HomeComponent.qml**
Die zentrale Home-Datei mit einer Auflösung von 320x480 Pixeln, die das Layout festlegt und mehrere Home-Buttons in sich zusammenfasst. In dieser Datei werden die durchzuführenden Aktionen als Reaktion auf die onClicked-Signals der einzelnen Buttons festgelegt.
- ▶ **HomeButton.qml**
Ist ein App-Icon bestehend aus einer Image- und einer MouseArea-Komponente. Properties: backgroundImage (string), enabled (bool)

DialerComponent QML-Ressources:

Der Dialer besteht aus einem Standard-Ziffernblock, einem Display zur Anzeige der getippten Nummer sowie einem Power-, einem Call- und einem Clear-Button.

Umgesetzt wurde der Dialer in vier QML-Dateien:

- ▶ **DialerComponent.qml**
Die zentrale Dialer-Datei mit einer Auflösung von 320x480 Pixeln, die das Layout festlegt und die einzelnen Komponenten in sich zusammenfasst. In dieser Datei werden die durchzuführenden Aktionen als Reaktion auf die onClicked-Signals der einzelnen Buttons festgelegt.
- ▶ **DialerDisplay.qml**
Ist die Nummernanzeige am oberen Rand des Dialers bestehend aus einer Image- und einer Text-Komponente. Properties: text (string)
- ▶ **DialerKey.qml**
Ist eine Zifferntaste des Ziffernblocks bestehend aus einer Image-, einer Text- und einer MouseArea-Komponente. Properties: text (string), fontsize (int), backgroundImage (string), icon (string)

AddressBookComponent QML-Ressources:

Das Adressbuch besteht aus drei Kontakt-Einträgen sowie einem (Dummy-)Button zum Anlegen eines neuen Kontakts.

Umgesetzt wurde das Adressbuch in zwei QML-Dateien:

- ▶ **AddressBookComponent.qml**
Die zentrale Home-Datei mit einer Auflösung von 320x480 Pixeln, die das Layout festlegt und mehrere AddressBook-Entries in sich zusammenfasst. In dieser Datei werden die durchzuführenden Aktionen als Reaktion auf die onClicked-Signals der einzelnen Buttons festgelegt.
- ▶ **AddressEntry.qml**
Ist ein Adressbucheintrag bestehend aus zwei Image-, und einer MouseArea-Komponente. Bei Selektion des Elements wird zwischen zwei Zuständen (front = Foto, back = Telefonnummer) gewechselt. Properties: image_front (string), image_back (string), currentImage (string)

Verbindung von QML und C++

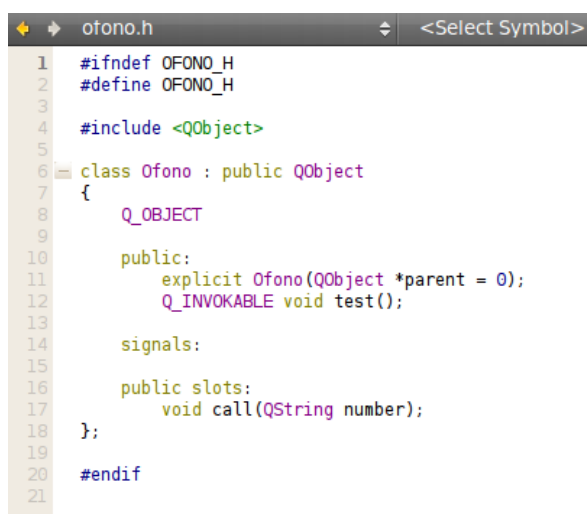
Um Methoden von C++-Objekten von QML-Files aus aufrufen zu können, sind folgende Schritte nötig:

in der Projektdatei mcm400-phone-app.pro:

```
QT += declarative
```

in der main-Methode:

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



```
1  #ifndef OFONO_H
2  #define OFONO_H
3
4  #include <QObject>
5
6  class Ofono : public QObject
7  {
8      Q_OBJECT
9
10     public:
11         explicit Ofono(QObject *parent = 0);
12         Q_INVOKABLE void test();
13
14     signals:
15
16     public slots:
17         void call(QString number);
18 };
19
20 #endif
21
```

Wichtigster Teil dabei ist der Aufruf der setContextProperty-Methode. Sie legt fest, welche Objekte in den QML-Ressourcen zur Verfügung stehen. Ein anderes Beispiel könnte etwa sein:

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

Werden an dieser Stelle eigene Objekttypen (ich habe hier ein Ofono-Objekt verwendet) übergeben, müssen diese von QObject abgeleitet sein und aufzurufende Signals/Slots definieren.

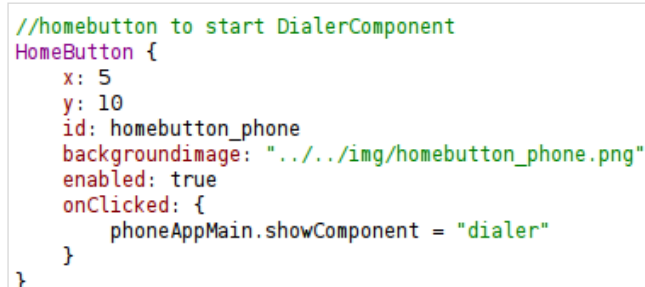
Es gibt zwei Möglichkeiten zur Definition von Slots:

- ▶ im public slots-Abschnitt der Klasse (zB void call(QString number))
- ▶ als Q_INVOKABLE im public-Abschnitt der Klasse (zB void test());

Signals werden im signal-Abschnitt der Klasse definiert.

Die definierten Signals/Slots können dann wie im ContextProperty festgelegt, von QML aus aufgerufen werden.

Hier ein Ausschnitt dazu aus der HomeComponent.qml-Datei:



```
//homebutton to start DialerComponent
HomeButton {
    x: 5
    y: 10
    id: homebutton_phone
    backgroundImage: "../img/homebutton_phone.png"
    enabled: true
    onClicked: {
        phoneAppMain.showComponent = "dialer"
    }
}
```