Project Documentation McmMOSLinuxPhone

David Berger, Paul Klingelhuber, Manuel Lachberger

MCM10 - WS2010/11

v0.1 -Dezember 4, 2010

Contents

1 The project

1.1 General

This Project was created in the Mobile Computing master's degree in the Mobile Operating Systems course. The goal for the project was to create a Qt application that acts as a mobile phone. With this application you can make and receive phone calls. The whole application was intended to run on an embedded Linux device, the Beagleboard (see section 1.2). We developed the application on a development PC (in our case it was Ubuntu 10.04) and used Ofono-phonesim (see section 1.3) as modem simulator.

1.2 Beagleboard

In our project we used the Beagleboard Rev C4 to actually run our application on a device.

The main features for the device (and the ones for us important) are:

- 720 MHz Omap3 CPU
- 256 MB RAM
- 256 MB NAND flash memory
- Audio/Video output
- a serial connection
- SD-Card slot
- USB-controller

We used the Angstrom embedded Linux distribution as base system and just extended this system with our own layer. This layer represents our Qt-application → our mobile phone.

1.3 Ofono-phonesim

To test the Qt-application we didn't want to flash it always on the mobile device. That's why we used a modem-simulator. In our case that was the ofono-phonesim application.

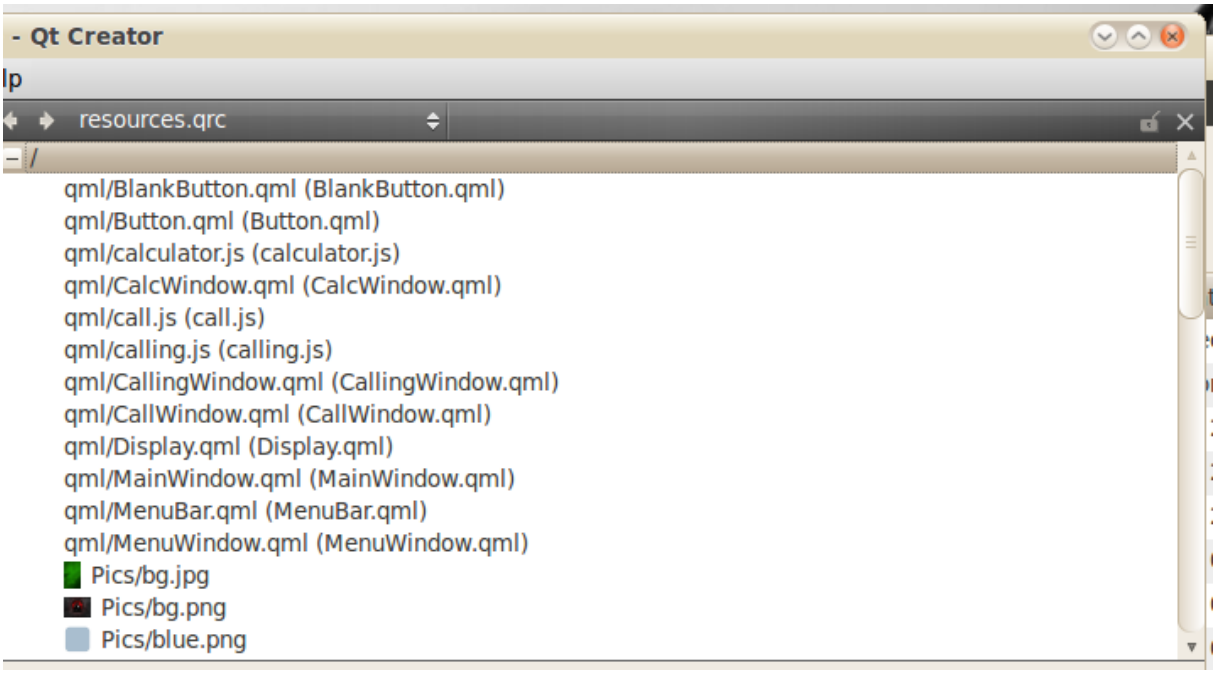
1.4 QML

To encapsulate the view from the application logic we decided to use QML. QML is an extension to the Qt-framework that is similar to CSS. With QML you have the advantage of creating/testing the view independent of the rest of the program. Additionally you can quite easily transfer data from the QML-part of the program to the C++-part with the signal-slot-concept of Qt. Furthermore it is possible to use JavaScript code inside the QML-part which makes the view quite dynamic easily changeable.

2 The application

2.1 General

In the following picture you can see the resources file that is required within Qt and contains all the required resources for our project.



2.2 The views

In our application there are 5 different views that are described in the following sections in detail which are triggered and changed with help of QML- transitions and state property change values.

2.2.1 The Home screen

In the following figure you can see the home screen. On the top of the screen there are 3 buttons:

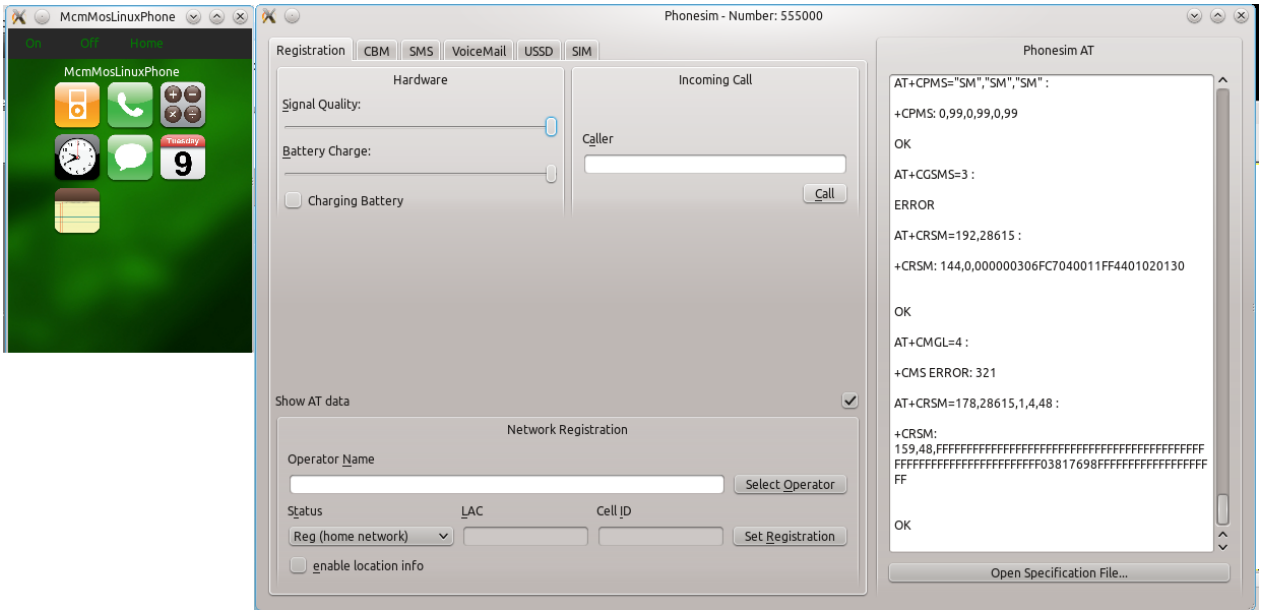
- On: turns on the modem functionality; right now it only sends the command to power on the ofono-phonesim simulator.
- Off: turns off the modem functionality; right now it only sends the command to power off the ofono-phonesim simulator.
- Home: switches from any other screen back to the main screen.

In the center of the screen there are the icons for the main functionality of the phone. Right now there is only the calling and the calculator implemented. But for the expandability and flexibility of the project there are additional icons available.

2.2.2 Power on

In the following figure you can see the view when the „On" button is pressed. Additionally here you can see the UI from the phonesim simulator. This UI is only visible when the phonesim simulator is started with the command:

```
#start ofono-phonesim with gui and specified port and xml-file
ofono-phonesim -p 12345 -gui moblin.xml
```



2.2.3 Power off

When pressing the „Off" button on the device the phonesim-simulator is killed and the application is returned to the home screen.

2.2.4 Calculator

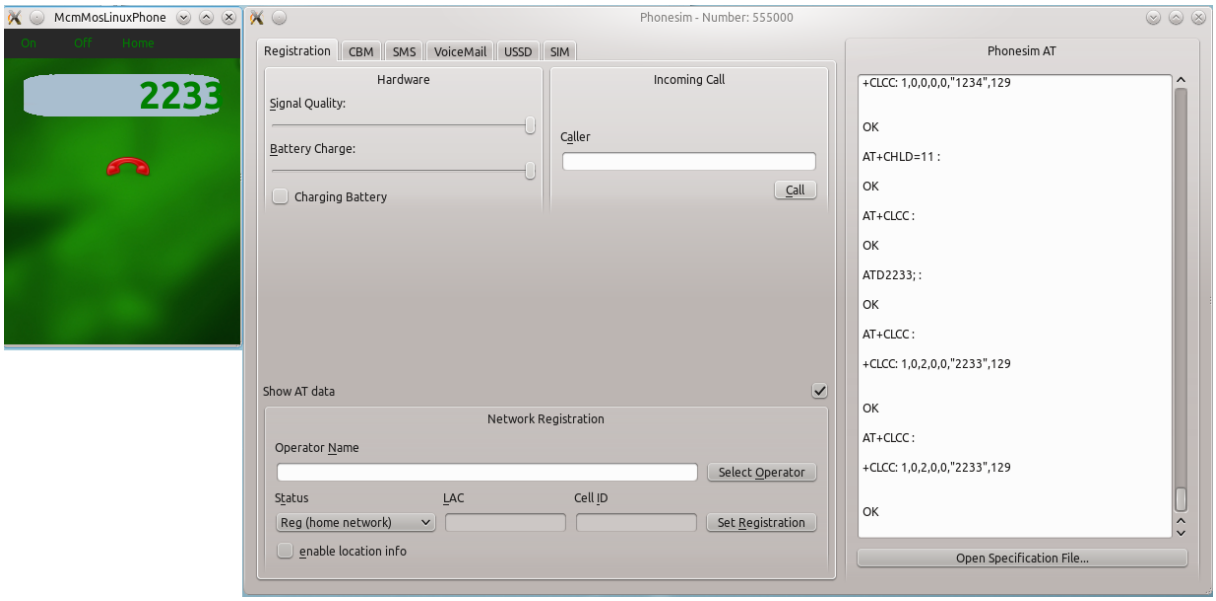
When activation the calculator you see the following screen, which lets you calculate things as a normal calculator. All operation are done via small single java script.

2.2.5 Outgoing Call screen

When the user wants to make a phone call he switches to the call screen which is visible in the following figure. Here the user can enter a number and then press on the green phone button which starts the phone call.

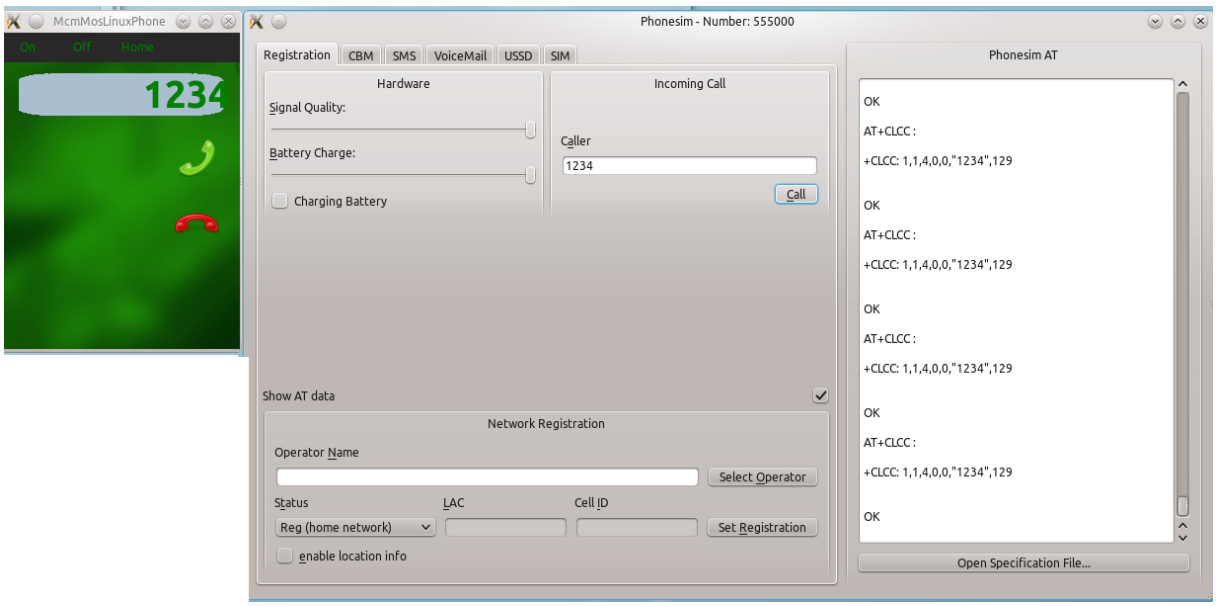
2.2.6 Outgoing Call Active

When the outgoing phone call is active the following screen is visible. Here we don't distinguish between a "ringing" call and an "active" call. It would be possible to distinguish between these too, e.g. to display the current active call duration, but that's not implemented right now.



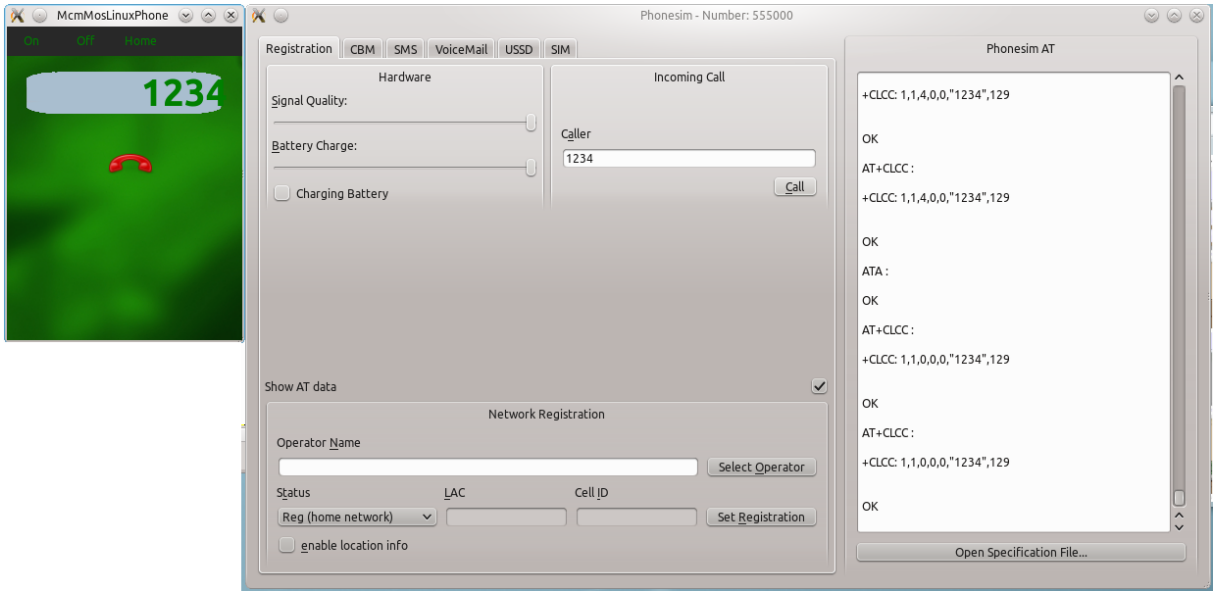
2.2.7 Incoming Call ringing

When there is an incoming call the following screen is displayed. The user can here decide if he wants to answer the phone call or reject it. Also he sees the caller's number at the top of the screen.



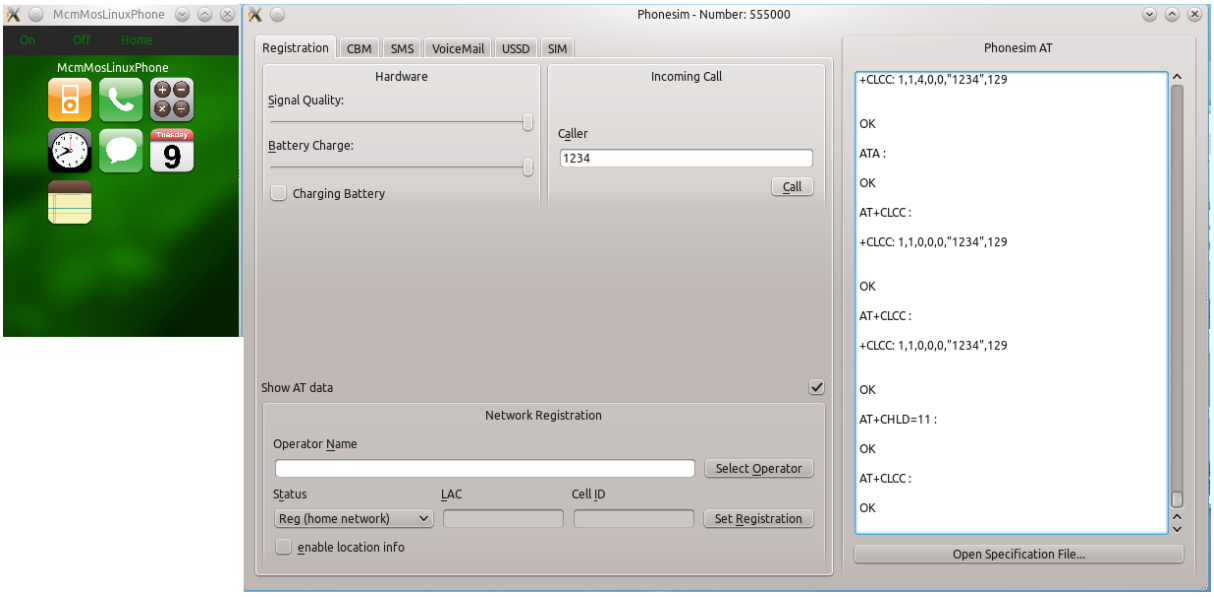
2.2.8 Incoming Call accepted

If the user decides to answer the phone call the application switches to the following screen.



2.2.9 Incoming Call ended

When the call is then ended the application terminates the phone call and switches back to the home screen.



2.3 Signal / Slots

To send comands from the QML-file to the Qt-C++-part, the well-known *Context* call (slots) have been used (e.g. setPowerOn()). When the app is started we load and initialize the *OfonoContext* (which represents the QML-part) in the Qt-C++-part of the application. When now emitting and receiving signals the *OfonoContext* (= QML) and our self-made class *Ofono* (= Qt-C++) are connected.

```
QmlApplicationViewer viewer;
viewer.rootContext()->setContextProperty("OfonoContext", new Ofono());
viewer.setSource(QUrl("qrc:/MainWindow.qml"));
```

In Ofono.h we have defined some calls we can use to communicate out of the QML-file with our layer:

```
public slots:
  void setPowerOn();
  void setPowerOff();
  void startPhoneCall(QString _number);
  void stopPhoneCall();
  void propertyChanged(const QString &_name,const QDBusVariant &_value);
  void answerCall();
```

Those methods could be called by using OfonoContext in qml, as we do in our button onClicked methods for example:

```
Button { width: 50; height: mainMenuBarID.height-1; id: onBtn; text: "On";  textColor:
"green";  onClicked:
            {
               callingWindowID.state = 'callingWindowStateOut'
               calcWindowID.state = 'calcWindowStateOut'
               callWindowID.state = 'callWindowStateOut'
               mainWindowID.state = 'mainWindowStateIn'
               OfonoContext.setPowerOn();
            }
      }
```

Once the Ofono contexts in qt-cpp wants to communicate with qml a simple signal is emited to the qml file with a signal defined in Ofono.h

```
signals:
  void incomingCall(QString id);
  void outgoingCall(QString id);
  void phoneCallAborted();
```

Those messages could be received by qml by using Connections and auto signal connections (incommingCall ... onIncommingCall):

```
Connections {
    target: OfonoContext
    onIncomingCall: {
```

```
        console.log("QML: Incoming Call: " + id);
        answerWindowID.state = 'answerWindowStateIn'
        answerWindowID.setNumber(id);
        callingWindowID.state = 'callingWindowStateOut'
        callingWindowID.setNumber(id);
        calcWindowID.state = 'calcWindowStateOut'
        callWindowID.state = 'callWindowStateOut'
        mainWindowID.state = 'mainWindowStateOut'
    }
    onOutgoingCall: {
        console.log("QML: Outgoing Call: " + id);
        callingWindowID.state = 'callingWindowStateIn'
        answerWindowID.state = 'answerWindowStateOut'
        callingWindowID.setNumber(id);
        calcWindowID.state = 'calcWindowStateOut'
        callWindowID.state = 'callWindowStateOut'
        mainWindowID.state = 'mainWindowStateOut'
    }

    onPhoneCallAborted: {

    }
}
```

As you can see, once an incoming call is triggered Ofono emits *incomingCall and onIncomingCall* is triggered in qml which changes states of our screens and uses javascript to set different kind of visuals.