



**Universität  
Zürich** <sup>UZH</sup>

# **Implementation of an automatic, on-demand Mobile Network Operator (MNO) selection mechanism on Android devices**

*Samuel Liniger  
Zürich, Switzerland  
Student ID: 08-913-832*

Supervisor: Christos Tsiaras, Daniel Dönni  
Date of Submission: August 23, 2013



---

## Abstract

In the mobile communication industry the call termination is considered to be a de facto monopoly, due to the fact that only the Mobile Network Operator (MNO) of the callee is able to terminate his calls [61].

To counteract this, the AbaCUS approach introduces an Auction-based Charging and User-centric System. AbaCUS assumes the existence of an automatic MNO selection mechanism that is implemented in the callee's smart-phone, which facilitates an on-demand MNO selection considering the termination rates [61] of all available MNOs.

This thesis describes a solution on Android based smart-phones, to select the MNO on-demand remotely in an automatic manner. Furthermore the proposed solution is measured by its efficiency in terms of energy, time and bandwidth consumption.



---

## Zusammenfassung

Die Anrufzustellung in der Mobilfunkbranche ist traditionellerweise ein de facto Monopol.

Diesem Zustand wirkt der AbaCUS-Ansatz (Auction-based Charging an User-centric System) entgegen, welcher eine auktionsbasierte Verrechnung und benutzerzentriertes System einführt [61]. AbaCUS setzt beim Smartphone des Empfängers einen Mechanismus voraus, der es erlaubt den Mobilfunkbetreiber über Remotezugriff anhand der Terminierungsgebühr auszuwählen.

Die folgende Arbeit beschreibt eine Lösung für Android-Smartphones, um den Mobilfunkbetreiber über einen Remotezugriff in einer automatischen Art und Weise zu wechseln. Des Weiteren wird die vorgeschlagene Lösung auf ihre Effizienz hinsichtlich der benötigten Batterie, benötigten Zeit und benötigten Bandbreite untersucht.



---

## Acknowledgments

I would like to thank Prof. Dr. Burkhard Stiller and the Communication Systems Group for providing this interesting Bachelor Thesis based on an ongoing research project.

Special thanks for the great supervision of Christos Tsiaras. Without the many e-mails as well as advice from discussions at his office this thesis would not have been possible. Additionally, I would like to thank Daniel Dönni and Marisa Bumbacher for proofreading this work.





---

## Table of Contents

<b>1 Introduction .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Description of Work .....	1
1.3 Thesis Outline .....	1
<b>2 Related Work .....</b>	<b>3</b>
2.1 Android .....	3
2.1.1 Software Architecture .....	3
2.1.2 Telephony Manager .....	4
2.1.3 Radio Interface Layer .....	4
2.2 Android Application Fundamentals .....	5
2.2.1 Android Activity .....	5
2.2.2 Android Services .....	6
2.2.3 Android Content Provider .....	6
2.2.4 Android Broadcast Receiver .....	6
2.2.5 Android Interface Definition Language (AIDL) .....	6
2.3 Development Prerequisite - Internal Android API .....	7
2.3.1 Accessing Internal Android API .....	7
2.4 Java Reflection .....	10
2.5 Google Cloud Messaging .....	10
2.6 AT Command Set .....	11
2.6.1 Test Commands .....	11
2.6.2 Read Commands .....	11
2.6.3 Set Commands .....	11
2.6.4 AT Commands in Practice .....	12
2.7 Auction- based Charging and User-centric System (AbaCUS) .....	13
2.8 Mobile/Wearable Device Electrosmog Reduction through Careful Network Selection .....	13
<b>3 Design .....</b>	<b>15</b>
3.1 MNO Selection Mechanism .....	15
3.2 AbaCUS Application .....	16
3.2.1 GSM Phone Service .....	16
3.2.2 AbaCUSApp .....	18
3.3 Limitations .....	20
<b>4 Evaluation .....</b>	<b>22</b>
4.1 Time Consumption .....	22
4.2 Power Consumption .....	26
4.3 Bandwidth Consumption .....	28
<b>5 Conclusions .....</b>	<b>29</b>
<b>6 Future Work .....</b>	<b>30</b>
<b>A : Abbreviations .....</b>	<b>35</b>
<b>B : Glossary .....</b>	<b>36</b>
<b>C : Tools and Environments .....</b>	<b>37</b>
<b>D : Power Consumption Calculation .....</b>	<b>39</b>
<b>E : AbaCUS Messages .....</b>	<b>40</b>
<b>F : Contents of the CD-ROM .....</b>	<b>43</b>

---

## **List of Figures**

1 Android Architecture Diagram [4] .....	3
2 Android RIL Architecture [51] .....	4
3 Activity Lifecycle [1] .....	5
4 Eclipse Access Rule .....	7
5 ADT Custom Access Rule .....	9
6 Android Custom Platform .....	10
7 AT+COPS=? Response .....	11
8 AT Commands Sent with puTTY .....	12
9 Key Elements of AbaCUS [61] .....	13
10 GSMPhone Instance .....	15
11 Select MNO Manually .....	15
12 Signing Application .....	16
13 Sending Application to Device .....	16
14 Initialize Phone Service .....	17
15 Expose Phone Service .....	17
16 Result Receiver .....	18
17 (a) AbaCUSApp QoS-C / TeR-C Chooser (b) AbaCUSApp Dialer .....	19
18 AbaCUS Request Service Message .....	20
19 MNO Switching Average Time on Mobile Scenarios .....	23
20 Min and Max Time on MNO Selection Scenarios .....	23
21 MNO Switching Average Time Signal Strength .....	24
22 MNO Switching Average Time .....	24
23 Large MNO Switching Time .....	25
24 Average Termination Time .....	25
25 Average MNO Scan Time .....	26
26 Power Consumption Calculation .....	27
27 AbaCUS Call Process .....	28

## **List of Tables**

1 Signal Strength .....	22
2 SGS2 Battery [28].....	27
3 MNO Switching Power Consumption .....	27
4 Available MNO List Power Consumption .....	27
5 Bandwidth Consumption .....	28

---

# 1 Introduction

The use of smartphones has been increasing in the last few years [40][65]. Many open standards and accessible Application Programming Interfaces (API) make it easier for developers to achieve their ideas and many communities, such as xda developers [64] or stackoverflow [55], provide good questions and answers concerning mobile application development. The question how to switch a Mobile Network Operator (MNO) programmatically on Android devices already arose in 2010 [41]. As far as it is known the answer of this question has not been published yet [42][50]. The main reason is that there is no method provided in the public Android API that allows to perform this task [47].

## 1.1 Motivation

To allow new MNO selection algorithms to reduce radiation [47], by choosing the MNO considering criteria such as the signal strength or to introduce new approaches for the call termination service, based on the MNOs call termination rates [1], it is mandatory to have a mechanism to switch the MNO programmatically. This means an application is needed which performs this task instead of the manual selection through the User Interface (UI).

## 1.2 Description of Work

This thesis provides a solution to select a MNO automatically in a background process, so that any other user interaction such as writing a message is not interrupted as long it is not dependent on a internet connection through the mobile network and that the selection can also be performed when the device is in stand-by mode. The mechanism is able to trigger the selection either from a remote location or from the device itself. The mechanism does not consider any security issues. Securing the MNO switching mechanism as well as authorization issues should be handled in the application layer. The main goal was to develop an automatic and on-demand MNO selection mechanism on Android platforms. This mechanism also has to be evaluated in terms of energy and time consumption during the MNO search and MNO selection procedure. Furthermore, the mechanism has also to be evaluated with respect to the bandwidth consumption during the MNO selection decision in the case that this happens remotely through AbaCUS.

The developed application is able to retrieve and store the user's location and to get a list with available networks at this location. Furthermore, it is able to send and receive AbaCUS messages, which was necessary for the evaluation of the bandwidth consumption.

In case that the primary goal of this thesis of a MNO selection mechanism would not be possible, an equivalent approach for WiFi networks should be examined. However, since a solution for the required mechanism has been deployed and evaluated, the secondary task to change WiFi Access Points (APs) instead of MNOs programmatically has not been investigated.

## 1.3 Thesis Outline

The structure of this thesis is the following: Section 2 provides information about related work with a strong focus on other attempts to answer the main question. Furthermore, solutions of similar problems are presented. The subsequent chapter develops the mechanism outlined in the Subsection 1.1 and shows assets, drawbacks as well as limitations of this work. The evaluation of the mechanism is discussed in Section 4 which is

---

followed by the Conclusions. The last chapter describes open questions and future work in the fields that have been discussed.

---

## 2 Related Work

This thesis is based on research interests in the field of the programmatical MNO switching on Android devices. The proposed solution makes use of existing approaches and extends them to select MNOs in an automatic and on-demand manner. The following passages introduce related work in this domain.

### 2.1 Android

Android is currently one of the most popular mobile platforms in the world [7]. Its openness and the publicly available source code was a major criterion to choose this platform for the purpose of this thesis. Nevertheless, it has to be kept in mind that Android is not entirely open [63]. In fact, it does not provide full access to the middleware and the application layer. In the kernel layer there are some components like power management, which developers are not allowed to extend or modify [9].

#### 2.1.1 Software Architecture

An important remark about the Android software architecture is that the Android Operating System (OS) consists of four levels. The lowest level is the system kernel, which is based on a Linux kernel that provides basic system services to upper layers, such as process isolation and scheduling, file system support, device drivers and networking [15]. In the middle of the four levels, native libraries such as databases or OpenGL are located. On the same level the Android runtime is located with the Dalvik Virtual Machine (DVM). DVM is a register-based VM, built-on the ARM architecture [17], which has been developed to allow very small but still high-performance implementations [46]. On top of this middle level there

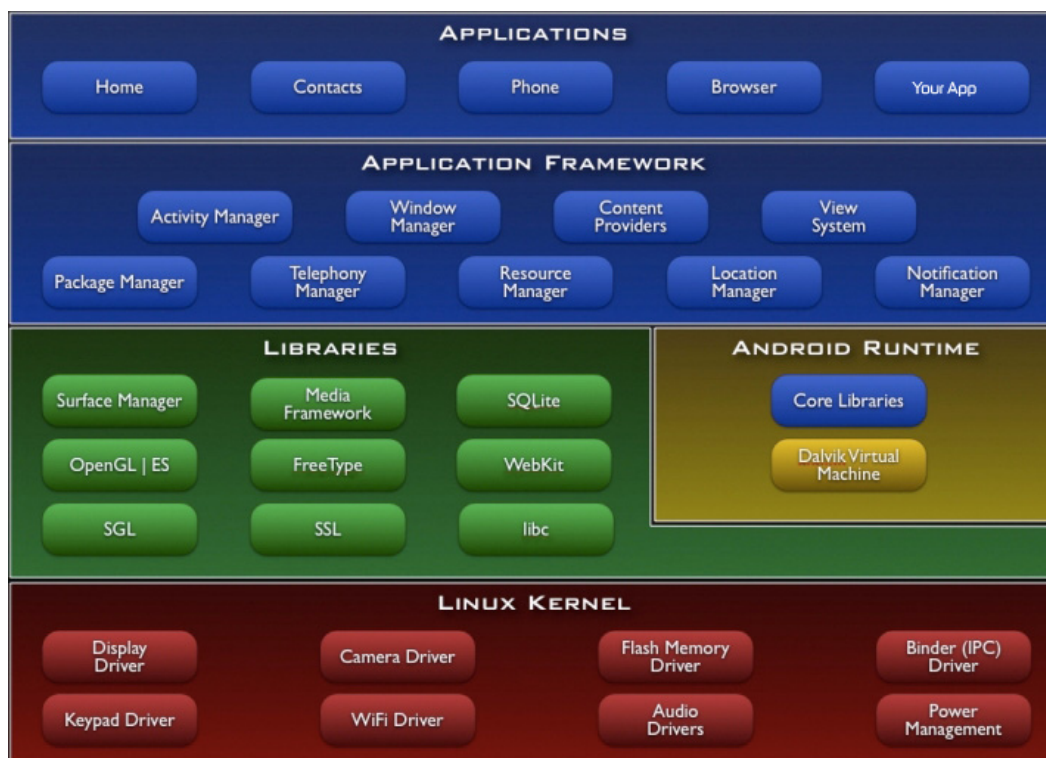


Figure 1: Android Architecture Diagram [4]

is the application framework which consists of several components. The Android software architecture as described above is shown in Figure 1.

On the top level there are user applications, as the provided ones by Google, or by the Android developers community. Google offers an accessible API as well as a Software Development Kit (SDK). From a modem integration's point of view the Telephony Manager is the most important component.

### 2.1.2 Telephony Manager

The Telephony Manager in this thesis point of view is the most important component, because it provides access to information about the telephony services on the device as well as access to certain types of subscriber information. Applications can register a listener to receive notification of telephony state changes [58]. The Telephony Manager is connected to the Radio Interface Layer (RIL) by using a socket connection. This socket is reserved only for this communication. Thus, it can neither be redirected nor expanded for other clients [63].

### 2.1.3 Radio Interface Layer

The RIL is a middle layer between the applications and the wireless module. It sends so called AT commands, which has been a standard way to access modems, to the baseband depending on the request of an application to control telephony services. Additionally, it reports AT responses from the baseband to the applications [51].

The RIL itself consists of two main parts, the Java part of the RIL (RILJ) and the native part (RILC). The RILJ is in the application framework layer and is the part that acts as a transmitter for the sockets mentioned in Subsection 2.1.2.

The RILC is part of the Hardware Abstraction Layer (HAL) and it is responsible to communicate with the RILJ. In more detail, it packages the corresponding AT command and sends it to the baseband. The RILC consists of three parts:

- RILD: This is a daemon process and communicates with the modem hardware through the Linux kernel modem hardware driver [63].
- Libril.so: A shared library mainly to complete the communication to the RILJ and to transmit messages to the Libreference\_ril.so [51].
- Libreference\_ril.so: Processes the communication with the RILD and dispatches calls to the RILD [51].

A better overview of RIL is illustrated on Figure 2.

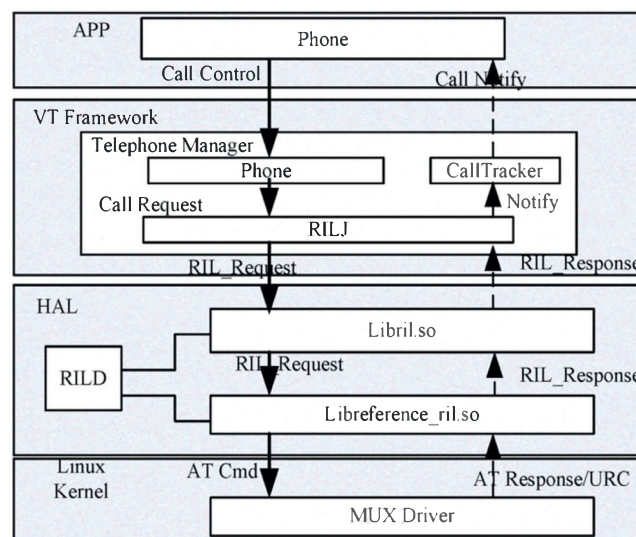


Figure 2: Android RIL Architecture [51]

---

So far, there is no published attempt that accesses the RIL in Android to change the MNO. In the stackoverflow community, the question how to get available networks using RIL, appeared [30] on 2013. One possibility stated is to implement one's own RIL. But there was no reply that documents this process yet.

## ***2.2 Android Application Fundamentals***

The Android applications are composed of one or more components, such as activities, services, content providers or broadcast receivers. These applications do not have one single entry point like traditional unix applications. Each component can act as a different entry point through which the system can enter the application and exists as its own entity [10]. Each component has a distinct purpose and a different lifecycle. These components do not have access to each other's memory space. To achieve Interprocess Communication (IPC) Android provides a tool called AIDL (Android Interface Definition Language).

### **2.2.1 Android Activity**

An activity is the only component with a user interface. It represents a single screen. The user interface is strictly separated from the code, thus it is defined in a layout file written in XML [35]. Each activity is separated from other activities. The communication takes place

with Intents. An Intent represents an abstract description of a function that one activity requires another activity to perform [35].

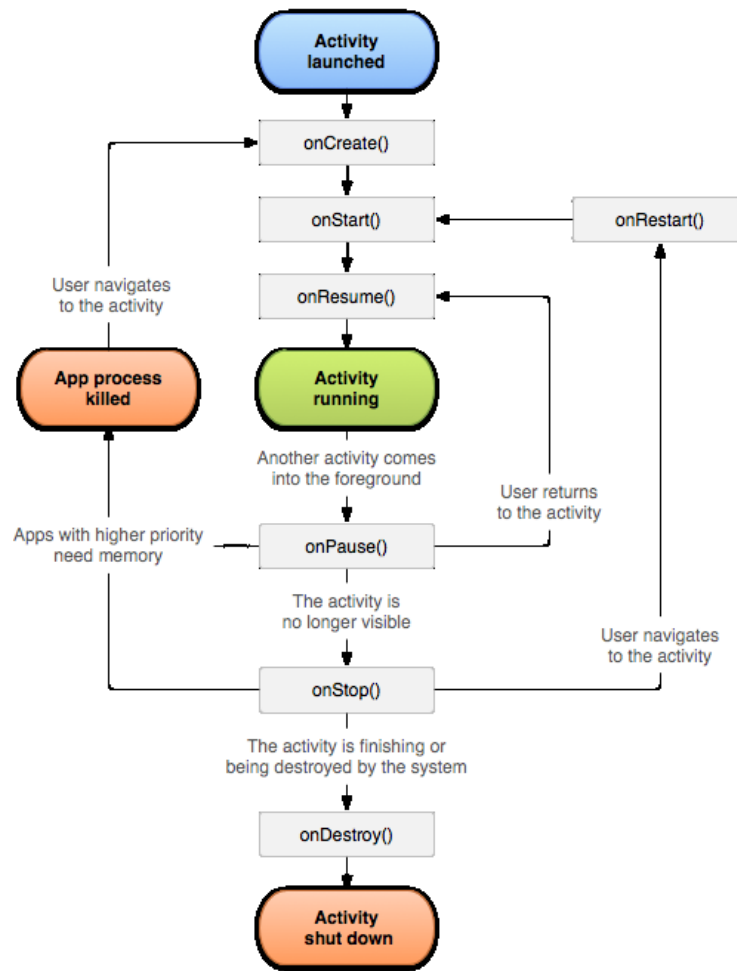


Figure 3: Activity Lifecycle [1]

The life cycle of an activity is the most complex one. Essentially it has the four states launched, running, killed and shut down [1], illustrated in Figure 3. There are three key loops within an activity. The entire lifetime happens between the first call of `onCreate(Bundle)`, where global resource are setup, to a single final call of `onDestroy()`, which releases the remaining resources. Then the visible lifetime of an activity that happens between a call to `onStart()` until a corresponding call to `onStop()`. In this phase the activity is visible, although it must not be necessarily in the foreground. That means that another activity that is visible can be in the foreground, but the other activity is still visible, when for example the foreground activity is transparent. The lifetime where the user interface of the activity is visible is called foreground lifetime. During this time the activity is in front of all other activities.

An activity can be destroyed if the activity is finished by calling `finish()` on it, or the system is destroying the instance temporarily to free memory.

### 2.2.2 Android Services

Services are running in the background to perform long-running operations or to perform work for remote processes. The Android platform avoids reclaiming service resources. That



---

means that once a service starts it is likely to be available unless memory gets scarce [35]. Other components can start services and let them run or interact with them.

### 2.2.3 Android Content Provider

A content provider manages persistent application data which can be stored in the file system, an SQLite [54] database on the web or any other persistent location. The content provider offers a mechanism that other applications can query or modify the data. Thus, an application that provides a `ContentProvider` can share data with other applications and manage the data model of an application [35].

### 2.2.4 Android Broadcast Receiver

A broadcast receiver is a component that can be registered to system wide broadcasts. These are system or application events. Example for system broadcasts are announcements that the screen has turned off or that the battery is low. There are also broadcasts initiated by applications, for example that data has been downloaded successfully and is available for use now. Broadcast receivers do not display a user interface, but they can create status bar notifications. Basically, a broadcast receiver acts as a gateway to other components [10].

### 2.2.5 Android Interface Definition Language (AIDL)

If developing Android applications, it might be necessary that different processes running in different applications have to communicate with each other. But one process cannot access the memory of another process in Android [5]. Therefore, the Android Interface Definition Language (AIDL) is provided, which defines the programming interface that the client and service can communicate using Interprocess Communication (IPC). For that, they both have to decompose their objects into primitives the OS can understand. Afterwards these objects are marshalled into code that is handled with AIDL. The AIDL interface has to be defined in an `.aidl` file which uses the Java programming language syntax. This file has to be stored in the `src/` directory of the project. The Android SDK tools generate an `IBinder` interface based on the `.aidl` file when the project is built. The service then has to implement this interface providing the methods, which the client applications may invoke when they are bound to the service.

## 2.3 Development Prerequisite - Internal Android API

Besides the public Android API that is accessible with the SDK, there is also an API that is not accessible via SDK which is located in the package `com.android.internal` [62]. In fact these methods are usable with Java reflection. However, there are advantages as well as drawbacks of using Java reflection.

In the software development of Android applications using Android SDK a jar file called `android.jar` is referenced. Thus it is added to the build path. This library has all classes from `com.android.internal` removed, as well as all classes, enumerations, fields and methods that are marked with the annotation `@hide`. If the application is launched on a device the library `framework.jar` is loaded. This library is equivalent to the library referenced in the SDK. However, there is one difference. The `framework.jar` library contains all internal API classes and all API components that are annotated with `@hide`. These classes and methods can be accessed with Java reflection.

---

### 2.3.1 Accessing Internal Android API

Accessing the internal Android API requires the `android.jar` to be replaced by the `framework.jar` to gain access to the internal classes. However, this is not immediately working. The reason for that is that the Android Developer Tools (ADT) plug-in for the Integrated Development Environment (IDE) Eclipse [24], which is providing a development environment for building Android applications, forbids the usage of anything of the package `com.android.internal` by adding an access rule to the Java Build Path, which needs to be removed.

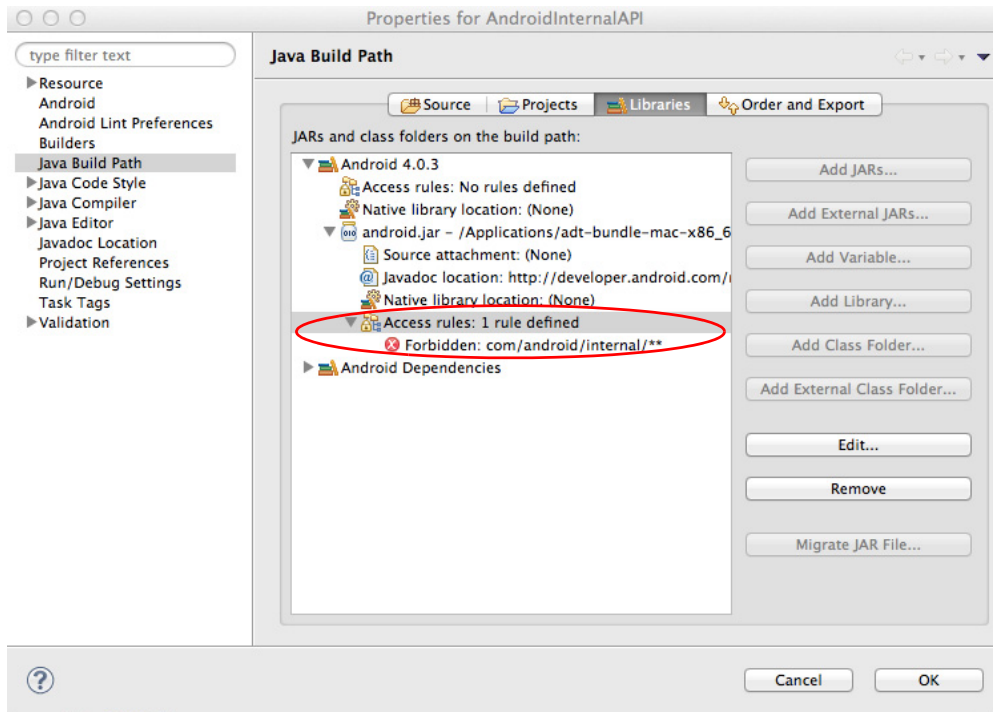


Figure 4: Eclipse Access Rule

A developer that needs to access anything from the internal API has to do the following steps: 1) Obtain Original Android Framework 2) Create Custom Framework 3) Modify Eclipse Access Rule.

#### 2.3.1.1 Obtain Original Android Framework

There are two different ways to obtain the original Android framework. One approach a developer could take would be to compile the framework himself, due to the fact that Android is an open source mobile OS [23]. However, there is another way by getting the runtime equivalent that is loaded on the device, which is located at `/system/framework/framework.jar`. Within this work the second approach has been chosen because this is less time consuming and a working solution has already been published. After it has been downloaded successfully it has to be extracted, for example by the command `jar xf framework.jar`. If the extracted folder does not contain a file `classes.dex` the file `/system/framework/framework.odex` has to be downloaded from the device. This file has to be disassembled with `baksmali.jar` [53] by the following command: `java -jar baksmali framework.odex`. If errors occur with the suggestion to download more odex files they have to be downloaded in the same location where `framework.odex` has been downloaded and the command has to be executed again. This will generate Android

---

platform related classes as smali files in a folder named `out`. Smali files are the disassembled Java classes in an editable form [13]. The folder `out` has to be assembled with `smali.jar` [53] by the command `java -jar smali out`. The assembled file is named `out.dex` and is equivalent to the file `classes.dex`, which has to be converted to a jar file using a tool called `dex2jar` [21] and then the resulting jar file has to be extracted. This can be done with the command `jar xf framework.jar`. The extracted folder contains all `.class` files of the package `com.android.internal` in the folder corresponding to the package name.

### 2.3.1.2 Create Custom Framework

To access the internal API in an IDE such as Eclipse, a custom framework has to be created which contains the classes and methods of the `internal` package. To create the custom framework, the Android SDK's `android.jar` has to be extracted first. This file is located at the Android SDK's installation folder in `SDK/platforms/android-X/android.jar`, where X is the API Level that is targeted to be customized such as level 15 for Android 4.0.4. The API level is an integer value to uniquely identify the framework API revision offered by a version of the Android platform [3]. All files that have been extracted from the original Android framework have to be copied in the previously extracted folder. Already existing files have to be replaced. Then all files in this folder have to be compressed again in `android.zip` followed by a renaming in `android.jar`. When the `android.jar` file will be added to the build path all the methods of the package `com.android.internal` will be accessible. The original Android framework library could either be replaced by the custom platform by replacing the original `android.jar` with the created one or the created framework could be added as new platform. To add a new platform the entire folder of the original platform has to be copied. Then the original `android.jar` has to be replaced with the custom one. To distinguish this custom framework from the original one a custom name and custom API level has to be provided by adapting the file `build.prop` in the platform folder. The value under the entry `ro.build.version.sdk` has to be replaced by a desired number, which represents the API level. The value `ro.build.version.release` could be expanded with `.extended` to indicate that this is a customized platform.

### 2.3.1.3 Modify Eclipse Access Rule

Finally, the last hurdle is to modify the Eclipse access rule that prohibits the use of the internal API. There are different possible ways to achieve this. The first approach is to modify the ADT source code and build it, which has not been investigated within this work. The easier way is to modify the ADT's bytecode. Therefore the contents of the file `com.android.ide.eclipse.adt*.jar` which is located in the folder `plugins` of the Eclipse installation have to be extracted. The `*` in the file name is a placeholder, hence the complete file name may differ depending on the ADT version. In other words, the contents have to be pulled out in a separate folder. In the subfolder `com/android/ide/eclipse/adt/internal/project` of the extracted folder the file `AndroidClasspathContainerInitializer.class` has to be opened in an editor that supports non-printable characters, for example `SlickEdit` [52] or `notepad++` [37]. The string `com/android/internal/` needs to be replaced with another string. In this work it has been changed to `com/android/internax/**`. When this file has been saved, the folder has to be compressed again with the same name as before. It has to be ensured, that the internal root folder of the archive is the same as the original, otherwise Eclipse will not recognize it. Finally, the archived folder has to be renamed to `*.jar`. The original ADT jar

file has to be replaced with the new one. After restarting eclipse the internal API is accessible. Another, less time consuming way, that worked successfully with ADT version 21 and 22, is to create a new access rule that allows to use classes out of the package **com/android/internal/\*\***. Because the access rule in the subentry **android.jar** cannot be modified, a new access rule should be created directly below the android platform. The resolution must be "Accessible" and a rule pattern has to be created (Figure 5).

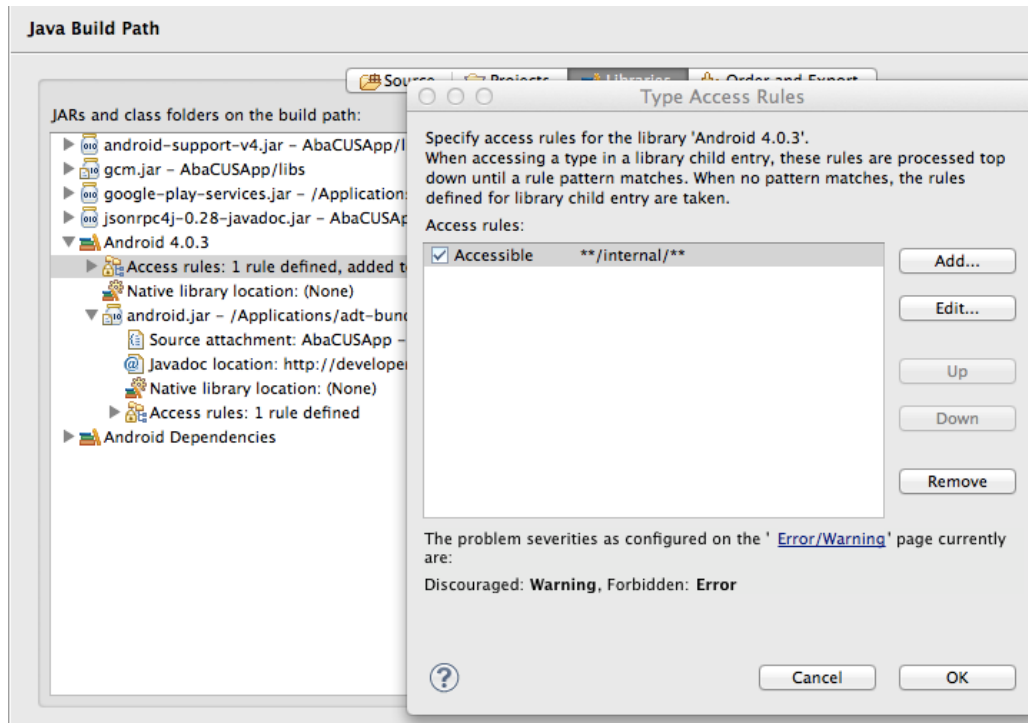


Figure 5: ADT Custom Access Rule

To make it possible to use the custom platform, the platform of the project has to be changed to the newly added one, as illustrated in Figure 6.

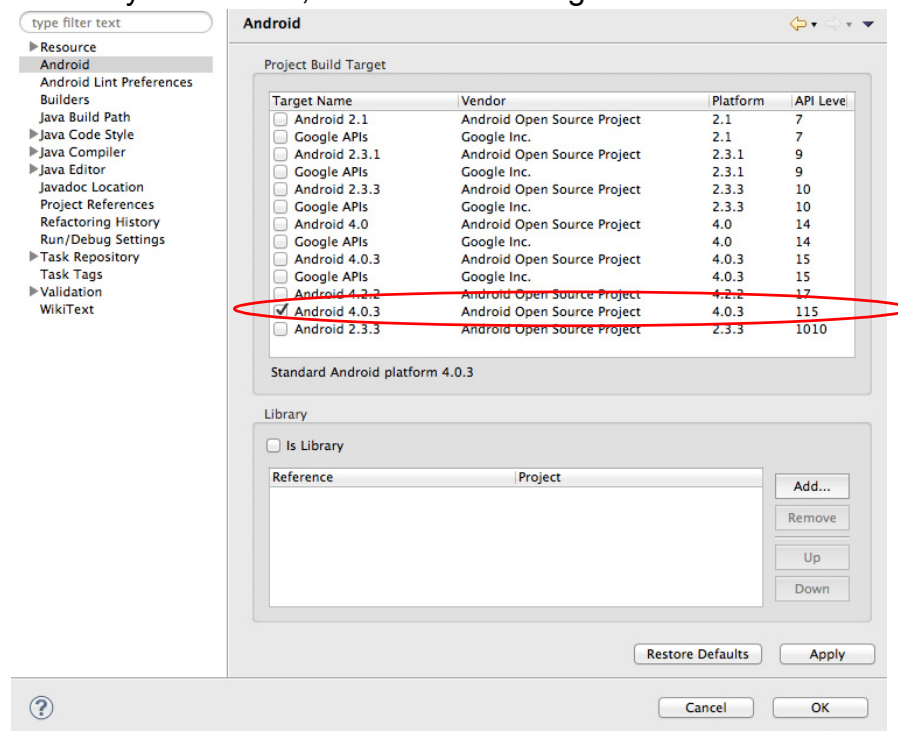


Figure 6: Android Custom Platform

## 2.4 Java Reflection

Reflection is a technique to examine or modify the runtime behavior of applications in the Java Virtual Machine [60]. It is very powerful, because it can enable applications to perform operations that otherwise would not be possible. For example an application can make use of external, user-defined classes or private members on classes can be examined. But reflection should not be used carelessly, because of three main drawbacks. There is a big performance overhead, since reflection involves types that are dynamically resolved. That is the reason why certain Java Virtual Machine optimizations can not be performed and the use of reflection results in slower performance. Therefore, reflection should not be used in sections of code which are called regularly. Another drawback is that reflection requires runtime permission. This may not be present when an object (security manager) exists, that defines a security policy for an application. Thus, if a security manager exists and reflection is used, a Security Exception would be thrown [59]. A big benefit of reflection is the possibility to access private fields and methods or fields and methods that are not accessible via SDK. On the other hand, this can result in unexpected side-effects such as non working code in future releases of the platform.

## 2.5 Google Cloud Messaging

Google provides a service called Google Cloud Messaging (GCM) for Android to send data from a server to a Android-powered device [29]. This service handles everything from queueing the messages to the delivery to the target Android application running. The messages can contain up to 4kb of payload.

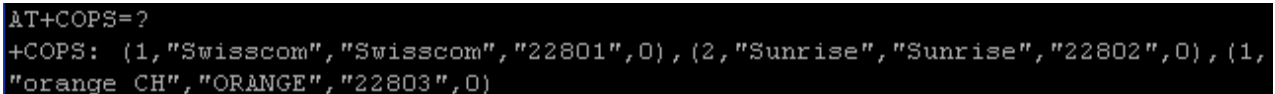
---

## 2.6 AT Command Set

The AT command interface has been a standard way to access modems as Computer peripherals [63]. Generally an AT command consists of three parts. It starts with **AT** followed by a command and ended with the line termination character [66]. Particularly there are three different types of AT commands which will be introduced in more detail in the following sections. All the commands discussed here are network service related and specific to GSM modem access.

### 2.6.1 Test Commands

Test commands test the existence of a command and check its range of subparameter(s) [66]. The format of those commands is **ATxxx=?**. To get a list of supported MNOs for example, the command **AT+COPS=?** has to be sent to the GSM modem. Figure 7 shows



```
AT+COPS=?
+COPS: (1, "Swisscom", "Swisscom", "22801", 0), (2, "Sunrise", "Sunrise", "22802", 0), (1,
"orange CH", "ORANGE", "22803", 0)
```

Figure 7: AT+COPS=? Response

the answer to this, where an integer is indicating the availability of the operator, out of four possible states:

- 0 unknown
- 1 available
- 2 current
- 3 forbidden

The next parameters are the long and short alphanumeric format of the name of the operator. The five digit number that is following represents the Mobile Country Code (MCC) which is three digits followed by the Mobile Network Code (MNC), which is the code for the network provider [44].

### 2.6.2 Read Commands

The read AT commands, as indicated by the name, read the current value of the subparameter(s) [66] and the format of such commands is **ATxxx?**. For example, to get the current registered network the command **AT+COPS?** has to be sent to the GSM modem.

### 2.6.3 Set Commands

To set new subparameter values, set commands have to be used. The AT command interpreter will return **OK** in the case that the command has been successful, otherwise an error or informative result code will be returned [66]. The format of set AT commands looks like this: **ATxxx=a,b**. An example related to the thesis is the command to set the MNO, which is the: **AT+COPS=1,2,"22801"** command. The first integer defines the mode, with five different values:

- 0 automatic
- 1 manual
- 2 deregister from network
- 3 set only
- 4 manual/automatic; if manual selection fails, automatic mode is entered

---

The second integer shows out of three possible values the format, how the MNO is referenced:

- 0 long format alphanumeric
- 1 short format alphanumeric
- 2 numeric

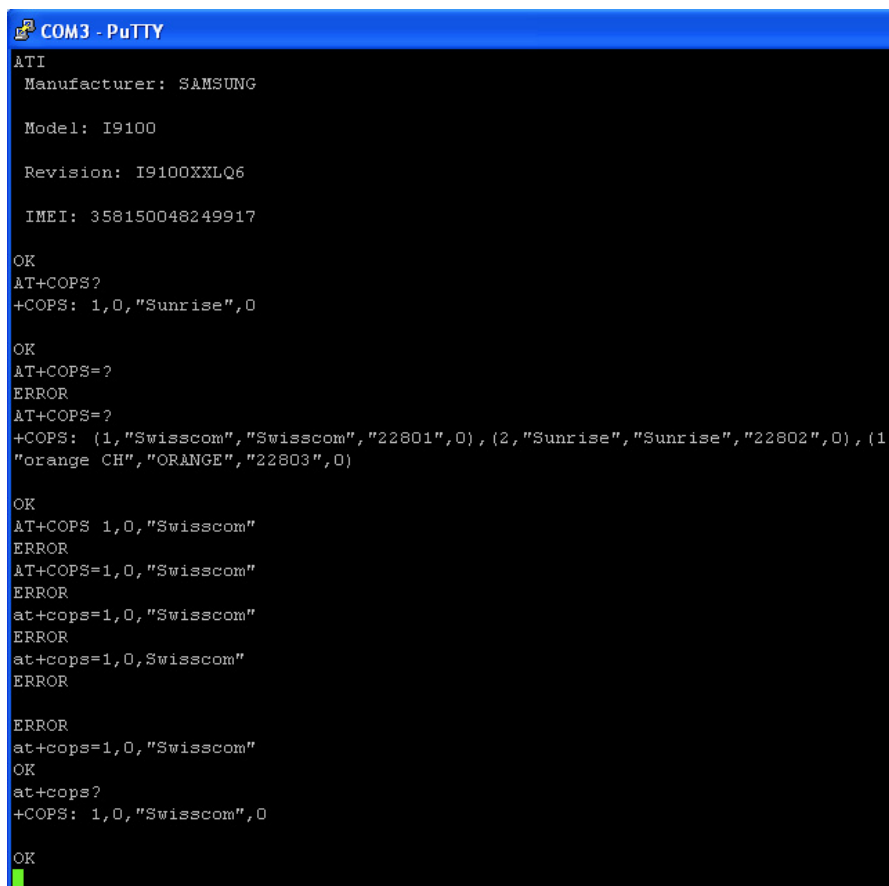
Thus if numeric format has been chosen, the last parameter that identifies the MNO has to be the Mobile Country Code plus the Mobile Network Code, as is evident in the example above. Otherwise if the format is alphanumeric the MNO is referenced by a name either with up to 16 characters (long) or 8 characters (short) [12].

#### 2.6.4 AT Commands in Practice

There have been many attempts to send AT commands to Android devices, either as peripheral from a computer or directly from the device itself [31]. But not all issues have been solved yet.

Within this thesis both attempts have been done to send AT commands as peripheral from a computer Samsung Galaxy S II (SGS2) as well as to send AT commands from this device itself.

The attempt to send AT commands from a computer with Windows XP was successful. Prerequisites were that the correct GSM modem driver of the used SGS2 was installed. Afterwards the modem could be addressed over the correct COM port with puTTY [11]. The outcome of it was a successful MNO selection which is illustrated in Figure 8.



```
COM3 - PuTTY
AT
Manufacturer: SAMSUNG

Model: I9100

Revision: I9100XXLQ6

IMEI: 358150048249917

OK
AT+COPS?
+COPS: 1,0,"Sunrise",0

OK
AT+COPS=?
ERROR
AT+COPS=?
+COPS: (1,"Swisscom","Swisscom","22801",0),(2,"Sunrise","Sunrise","22802",0),(1,
"orange CH","ORANGE","22803",0)

OK
AT+COPS 1,0,"Swisscom"
ERROR
AT+COPS=1,0,"Swisscom"
ERROR
at+cops=1,0,"Swisscom"
ERROR
at+cops=1,0,Swisscom"
ERROR

ERROR
at+cops=1,0,"Swisscom"
OK
at+cops?
+COPS: 1,0,"Swisscom",0

OK
```

Figure 8: AT Commands Sent with puTTY

In contrast, the approach to send AT commands from Mac OSX according to [48] failed.



The next approach was to send AT commands from the device itself. A possible approach is also documented on the internet, but as far as it is known there is no published working solution yet [31].

## 2.7 Auction-based Charging and User-centric System (AbaCUS)

An example of a system that requires a mechanism as the one proposed in this thesis is presented by C. Tsiaras and B. Stiller at [61]. The authors propose a solution to counteract the monopoly of the call termination service in mobile networks. Therefore they introduce a system where the caller, who is the paying party, can influence which MNO will terminate the call. For that purpose the caller selects a defined Quality-of-Service class (QoS-C) and the Termination Rate Class (TeR-C). The QoS classes contain parameters related to sound quality and the network-access waiting-time. The TeR-Cs define the potential start-up cost and the desired charging rate. To perform a call, the caller sends a request to place a call to the Auction Authority ( $Au^2$ ). This request includes the Mobile Subscriber Integrated Services Digital Network Number (MSISDN) of the callee and the desired QoS-C as well as the TeR-C tolerance of the caller. Multiple MNOs can participate in an auction performed by the  $Au^2$  by sending the selected TeR-C preference per QoS-C. After the auction has taken place, the  $Au^2$  sends the information of the winning MNO to the device of the caller and the callee, to register to the winning MNO and terminate the call. This process requires an automatic and on-demand MNO selection mechanism. Figure 9 illustrates the key elements of AbaCUS.

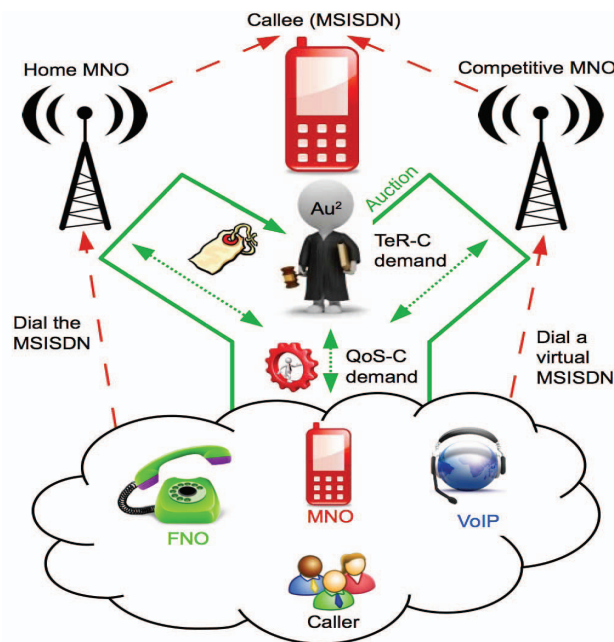


Figure 9: Key Elements of AbaCUS [61]

## 2.8 Mobile/Wearable Device Electromog Reduction through Careful Network Selection

Another work that could make use of an automatic and on-demand network selection is proposed by Seigneur et. al. [47]. The authors provide a network switching selection model and its algorithms that minimize the non-ionizing radiation of the devices during use. Their goal is to minimize the exposure of the mobile user to electromagnetic radiations while still



---

providing a certain Quality-of-Service (QoS). Within a proof of concept they validated the model and its algorithms. Due to the fact that the Android API does not provide a mechanism to force switching from one to another mobile network, the user has to manually select a network. This takes a lot of time because the provided mechanism by Android searches first for all available networks in any case. To avoid this, the proposed selection mechanism of this thesis can be used for the non-ionizing radiation minimization purpose.

---

## 3 Design

Based on current research and published solutions in the field of the thesis topic, the mechanism to select MNOs automatically and on-demand has been implemented. Furthermore, it is essential to list potential limitations and drawbacks of the mechanism.

### 3.1 MNO Selection Mechanism

Although the Android API does not provide any method to change the MNO, studying the Android 4.0.4 source code [6] the class **GSMPhone** was found. This class contains a public method **selectNetworkManually(...)**. This is part of the **com.android.internal** package and could therefore be used with the method described in Subsection 2.4. The class **PhoneFactory** provides methods to get different kinds of phone objects. To instantiate a **GSMPhone** object the method **getGsmPhone()** : **Phone** has to be invoked [16].

```
Context context = getApplicationContext();
ClassLoader cl = context.getClassLoader();
Class<?> PhoneFactory;
try {
    PhoneFactory = cl.loadClass("com.android.internal.telephony.PhoneFactory");
    Method get = PhoneFactory.getMethod("getGsmPhone", (Class[]) null);
    GSMPhone gsmPhone = (GSMPhone) get.invoke(null, (Object[]) null);
} catch (Exception e) {
    //exception handling
}
```

Figure 10: GSMPhone Instance

Afterwards, the method **selectNetworkManually(...)** can be invoked by the **GSMPhone** object with the required parameters **OperatorInfo** and a **Message**. **OperatorInfo** contains the information about the MNO to select. This includes the operator information as alpha numeric long, alpha numeric short and numeric. The different states that can be sent are **UNKNOWN**, **AVAILABLE**, **CURRENT** and **FORBIDDEN**. Within this work a selection could only be performed when a new **OperatorInfo** object with a correct MNO as numeric was created. The other values can be null or empty, as well as the state that the mechanism is working. The handler is used to schedule the message when the MNO has been selected successfully. Therefore it has to obtain the message: **EVENT\_NETWORK\_SELECTION\_DONE**.

```
int EVENT_NETWORK_SELECTION_DONE = 200;
OperatorInfo operatorInfo= new OperatorInfo("Swisscom", "", "22801", "UNKNOWN");
Handler mHandler = new Handler();
Message message = mHandler.obtainMessage(EVENT_NETWORK_SELECTION_DONE);
gsmPhone.selectNetworkManually(operatorInfo,message);
```

Figure 11: Select MNO Manually

Before this mechanism is usable, two further things have to be done 1) run the application with a different shared user ID and 2) run the application under the phone process. To prevent a **SecurityException**, that is thrown when protected intents are sent by the methods wanted to be invoked, the application has to run either with the system user ID **android:sharedUserId="android.uid.system"** or the shared user ID

---

`android:sharedUserId="android.uid.phone"` [16]. This has to be set in the "AndroidManifest.xml" within the manifest-tag. In addition, the application has to run in the process `android:process="com.android.phone"`, so that the invocation of `getGsmPhone()` is allowed. This attribute has to be added in the tag of the component that should run in the phone process, for example in the activity-tag.

Because the shared user ID is used by more than one applications, all have to be signed with the same certificate [34]. Thus the application has to be signed with the system signature key. The way to get such a key is to run a custom rom, for example CyanogenMod, which also provides these certificates [20]. A custom ROM is a fully standalone version of the OS. Since Android is open source, developers are free to take the version of the phone's operating system that comes with your phone when you buy it (stock ROM) and optimize, modify or add things [19].

The process of signing an application is explained according to [8]. First of all, the application has to be exported as unsigned application package. The `platform.x509.pem` and `platform.pk8` have to be downloaded. These files have to be put into the same folder as the application to sign. With the tool `jarsigner` and the following command the application will be signed (Figure 12).

```
java -jar signapk.jar platform.x509.pem platform.pk8 Application.apk signedApp.apk
```

*Figure 12: Signing Application*

To get this application on the device, the partition has to be remounted read-write first. For that purpose superuser rights are needed. Afterwards the application can be sent to the device (Figure 13).

```
adb shell mount -o remount,rw /system
adb push signedApp.apk /system/app/Application.apk
```

*Figure 13: Sending Application to Device*

## 3.2 AbaCUS Application

The requirements for the application that handles a call in AbaCUS were that it is able to perform the MNO switching in an automatic manner. Additionally, the application has to store the current location and get the current available MNOs. During the MNO search there should not be any user interaction. To achieve a call, the application has to be able to dial an MSISDN and exchange all the AbaCUS messages. The final implementation consists of two applications. One is responsible for the interaction with the GSM modem to handle the MNO switching and MNO scanning mechanism. The other application performs the communication to the Au<sup>2</sup> server and provides a GUI to choose the TeR-C and QoS-C, a dialer to choose the callee and to store settings, such as the interval time to update the Au<sup>2</sup> server. The reason why the MNO selection mechanism is in a different application is, that this part runs with `android:sharedUserId="android.uid.system"` but the GCM service which is responsible for the Au<sup>2</sup> server responses, cannot run with this user id.

### 3.2.1 GSM Phone Service

The service `PhoneService` consists of one class providing the mechanism to access the `com.android.internal.telephony.ITelephony.gsm.GSMPhone`.

---

It implements the **IServicePhone** which is the interface providing the following IPC methods with AIDL:

- **void selectMnoNumeric(in int numeric)**
- **void selectMnoAlphaLong(in String alphaLong)**
- **String[] getOperatorList()**
- **void searchOperators()**

The two methods to select a new MNO triggers the selection. The method to get the operator list returns a list with the available MNOs that have been found after the invocation of the method **searchOperators()**. The selection process and the search process are asynchronous. Therefore a listener has been implemented that is called when the MNO has been selected successfully or if the available networks have been scanned. To send this information to another application using this service, a **ResultReceiver** has been implemented. This generic interface is used that the application which uses the **PhoneService** can receive a callback result from it [43].

On creation of the service, the **GSMPhone** object is instantiated with Java reflection. Afterwards, a phone state listener is started for notification purposes when the phone state has changed. Then it searches the available MNOs for the first time. After the service has been created and the start command was called, the **ResultReceiver** is created, that sends the callback results to the application using this service. Because the service should run until it is stopped explicitly for arbitrary periods of time this method returns **START\_STICKY**. This constant indicates that if the service's process is killed while it is started, the system will try later to re-create it [49]. Figure 14 shows how the service has to be instantiated from another application.

```
MnoSelectionServiceConnection connection = new MnoSelectionServiceConnection();
Intent intent = new Intent("com.example.phoneservice");
ResultReceiver resultReceiver = new PhoneServiceResultReceiver(new Handler());
intent.putExtra("receiver", resultReceiver);
boolean ret = bindService(intent, connection, Context.BIND_AUTO_CREATE);
startService(intent);
```

*Figure 14: Initialize Phone Service*

The class **MnoSelectionServiceConnection** (Figure 15) implements **ServiceConnection** and is used to expose the **PhoneService** object to invoke the provided methods.

```
public class MnoSelectionServiceConnection implements ServiceConnection {

    @Override
    public void onServiceConnected(ComponentName name, IBinder boundService) {
        InitService.setPhoneService(IServicePhone.Stub.asInterface((IBinder) bound-
        Service));
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        InitService.setPhoneService(null);
    }
}
```

*Figure 15: Expose Phone Service*

---

The **PhoneServiceResultReceiver** class (Figure 16) is used to invoke methods of the activity using the service. If a registration to a MNO was successful the **PhoneServiceResultReceiver** gets the MNO numeric name. When the service has scanned available networks successfully, a string array with the available networks is returned with the MNO numeric and MNO alphanumeric long in the following format: **Numeric;Alphanumeric Long**, for example **22801;Swisscom**.

```
public class PhoneServiceResultReceiver extends ResultReceiver {
    public static final int RESULT_SCANNED = 0;
    public static final int RECEIVE_STATE_IN_SERVICE = 1;

    public PhoneServiceResultReceiver(Handler handler) {
        super(handler);
    }

    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
        switch (resultCode) {
            case RESULT_SCANNED:
                String[] operators = resultData.getStringArray("op");
                break;
            case RECEIVE_STATE_IN_SERVICE:
                String currentOperator= resultData.getString("mno");
                break;
            default:
                break;
        }
    }
}
```

*Figure 16: Result Receiver*

### 3.2.2 AbaCUSApp

The part of AbaCUS on the Android mobile device is called AbaCUSApp. This application is responsible for the creation of the **PhoneService** and the communication to the Au<sup>2</sup> server. The developed application within this thesis is a simple prototype which provides the mechanism to illustrate the process of a call in AbaCUS. It consists of several services and activities running either in the background or providing a GUI. The main part is a service called **InitService** which creates the **PhoneService**, register the device for GCM and initializes the **CalleeData** of the device. This service is always called on device boot-up. If the application starts for the first time and the MSISDN of the inserted SIM card can not be extracted, a preference screen appears to insert the MSISDN manually. Additionally the time interval to send updates to the Au<sup>2</sup> server can be inserted in milliseconds. These updates include the current location. The GUI of AbaCUSApp consists of four different activities. When the application is started, a selection matrix appears to choose the TeR-C and QoS-C where the rows are related to the TeR-C and the columns to the QoS-C. This selection happens in a matrix (Figure 17), because it allows the choice to be made by one click.

If these parameters have been chosen, the next screen appears which provides a dialer function. A text field expects the MSISDN of the callee. Additionally, a contact stored on the device could be chosen by clicking the contact button. If the callee's MSISDN is entered in the text field, the call can be triggered by the dial button (Figure 17). If the Menu button of the device is pushed in any of the two introduced activities, a screen to change settings appear. If the Menu button is pressed in the activity of the QoS-C and TeR-C chooser there

are different options available. One possibility is that the preference screen that has been introduced before will open. Two buttons are related to the GCM service. The one is to unregister the push notifications and the other to register it. The server name of the GCM service is hardcoded and not configurable in this prototype.

Then there is the possibility to see the current available MNOs by clicking **Available Operators**. There is also a button that opens a browser and shows the AbaCUS website if clicked.

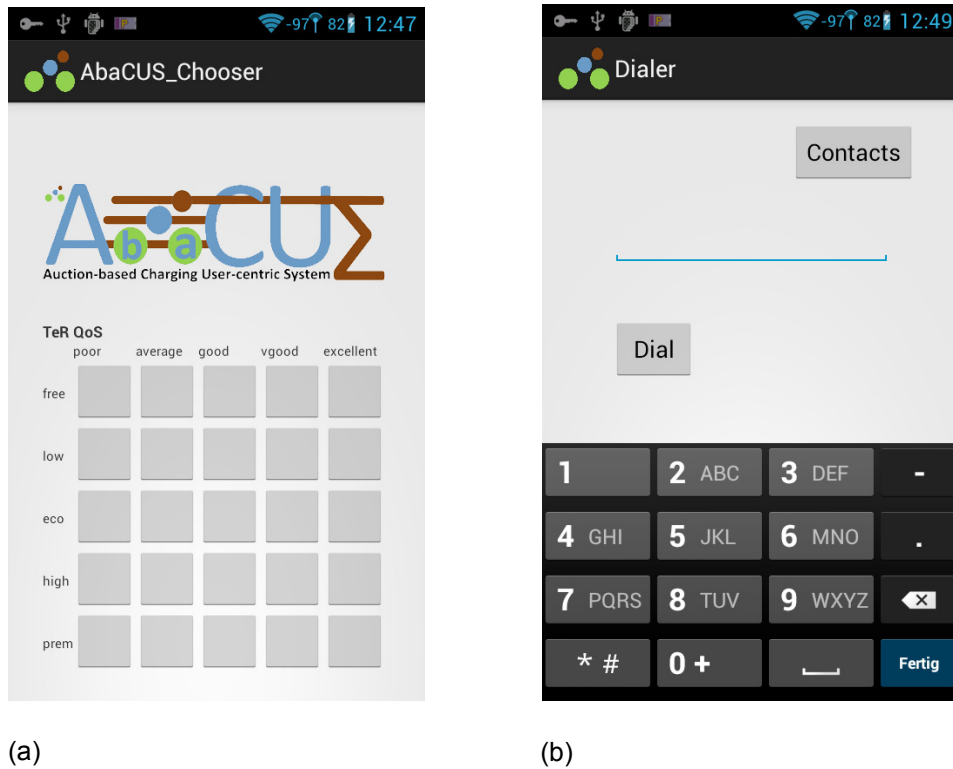


Figure 17: (a) AbaCUSApp QoS-C / TeR-C Chooser (b) AbaCUSApp Dialer

### 3.2.2.1 AbaCUS Messages

To send AbaCUS Messages there are two communication channels. The messages from the Au<sup>2</sup> server to the device are sent with GCM. The reason of this is that the device has to act as a server to get messages. But if a persistent listener would be implemented, this would have a big impact to the battery life. GCM instead provides an already optimized mechanism which can send up to 4kb of payload. If a push notification arrives at the device it extracts the payload. This data is always a key value pair. If the mobile device, that should receive a notification, is not connected to the server but registered, it gets the message the next time it is online.

Upstream messages to the server are sent through an URL connection as JavaScript Object Notation (JSON). JSON is a lightweight text-data interchange format [33]. That is the reason why this syntax has been chosen. The following messages have been implemented:

- requestService (Figure 18)
- register
- update
- ok

---

A general AbaCUS message always begins with an item called message type. This indicates which kind of message it represents, for the server to know how to parse it.

```
{
  "msgType":0,
  "qos":"GOOD",
  "time":1376064992915,
  "msisdn":"00306977990055",
  "imei":"358150048251046",
  "sim":"89300100090303909590",
  "uuid":"0019cfe74d286e",
  "calleeMsisdn":"00 372 5929 0600",
  "long":8.5723798,
  "key":["C@410ef850",
  "ter":"ECONOMY",
  "acc":1138,
  "lat":47.4389765
}
```

*Figure 18: AbaCUS Request Service Message*

Four message types have been implemented which are mapped to an integer as follows:

- Message Register = 0
- Message Update = 1
- Message Request Service = 2
- Message Status = 3

The register method needs the **CalleeData** (of the device) as parameter. The callee data consists of the IMEI, the SIM Card Number, MSISDN, UUID, current location and a key. An update message consists only of the current location. Update messages are sent in a predefined time interval to keep the Au<sup>2</sup> up to date. If a device requests a service it sends the current position, the desired QoS-C, the selected TeR-C, **CalleeData** of the caller, the MSISDN of the one to call and a generated key. Possible status messages after a register message has been sent are **OK** or **ERROR**. Other status messages are **OK**, **ERROR**, **POSTPONE** in the case an update message has been sent. In the case that the device sent a request service, possible status responses of the server are **BUSY**, **AWAY** or **ERROR**. To ensure that the application can be extended, the Au<sup>2</sup> client holds only the methods of the available AbaCUS messages. If such a method is invoked, it opens a generic JSON client which takes JSON Objects that are sent to the servlet of the Au<sup>2</sup> server. The structure of each message with its attributes is in Appendix E.

### 3.3 Limitations

There are different limitations on the developed solution. The mechanism works on a SGS2 with CyanogenMod 9. The attempt to run it with a HTC Desire Z on CyanogenMod 7 has been done. But this is not possible without adapting the code. For example the method **selectNetworkManually** does not expect a parameter called **OperatorInfo**, instead it is called **NetworkInfo** in Android 2.3.7. Once this has been adapted the selection mechanism has been working. The reason why the mechanism selection is working only on a custom ROM is that the **PhoneService** has to run with **android:sharedUserId="android.uid.system"** or **android:sharedUserId="android.uid.phone"**. Such an application can only be installed if it is signed with the system signature, thus the same signature the firmware has been signed with. To get the application on a device it must be rooted so that the partition

---

where to place the application can be remounted with writing rights. Otherwise the application cannot be installed.

Other limitations are that no security issues have been considered. The dispatched AbaCUS messages are not encrypted but sent in plain text. Thus a potential attacker could observe these messages.



---

## 4 Evaluation

To examine if the implemented mechanism is practicable, an evaluation with respect to time consumption, power consumption and bandwidth consumption, in case of AbaCUS, has been performed. Time consumption and power consumption were tested together for the MNO selection mechanism and the mechanism to get available networks. However, during the evaluation a few parameters could not have been controlled. Two foreign SIM cards were used [27][39] where no guarantee was given that the registration process to a Swiss MNO was performed with high priority. Besides this, the tests were not performed under laboratory conditions. The signal strength sometimes has been unstable and has not been under control.

### 4.1 Time Consumption

To test the time and power consumption, a test has been implemented to switch between three available Swiss operators (Orange [38], Sunrise [56] and Swisscom [57]). To get a complete test, the switching took place in all directions, thus one test step consisted of six switches. This test was repeated a hundred times which led to total 600 hops in total. In the end, the average time needed per case and the time needed for the whole test was calculated. In each cellular system for mobile communications transmitters, typically base stations, cover a certain area, a cell [44]. These cells have a unique 16 bit cell identifier [45]. After a successful hop, the signal strength and the corresponding cell id have been measured five times. The cell id and corresponding signal strength have been measured the first time immediately after the hop, then after every 500 ms. The reason why the signal strength has been captured five times is that it is apparent if the signal strength is stable or if it signals off. The corresponding cell id has been tracked to make sure that a possible signal strength oscillation is not due to a change to a different cell. Valid signal strength values are 0-31 and 99. The relation between these values and the signal strength in dBm is illustrated in Table 1 [22].

Table 1: Signal Strength

value	dBm
0	-113 dBm or less
1	-111 dBm
2 - 30	-109 dBm -53 dBm
31	-51 dBm or greater
99	not known or not detectable

This experiment has been executed six times during a weekday on the following time frames (08:00, 10:30, 12:00, 14:00, 17:00, 02:00). These time frames have been selected so that the measurements concluded are spread during the day when the network state (e.g, network load) changes between the rush hours. In a first step, the phone was connected to power supply, resting at the same location. The next time the experiment was repeated one more time without the power supply to measure the battery consumption. Then the experiment has also been executed when moving in a train from Zurich airport to the countryside of Lucerne. The results of the MNO switching time are shown in Figure 19. The data for the bars of the MNO selection while stable consists of total 6 times 100 hops

collected in different hours. In contrary the data of the MNO selection while moving consists of total 100 hops. The last two bars show the mean over all cases. Thus the mean of the MNO selection while stable is calculated out of total 3600 hops. In contrary the mean of the other case is calculated of the 600 hops of the MNO selection evaluation while moving. One

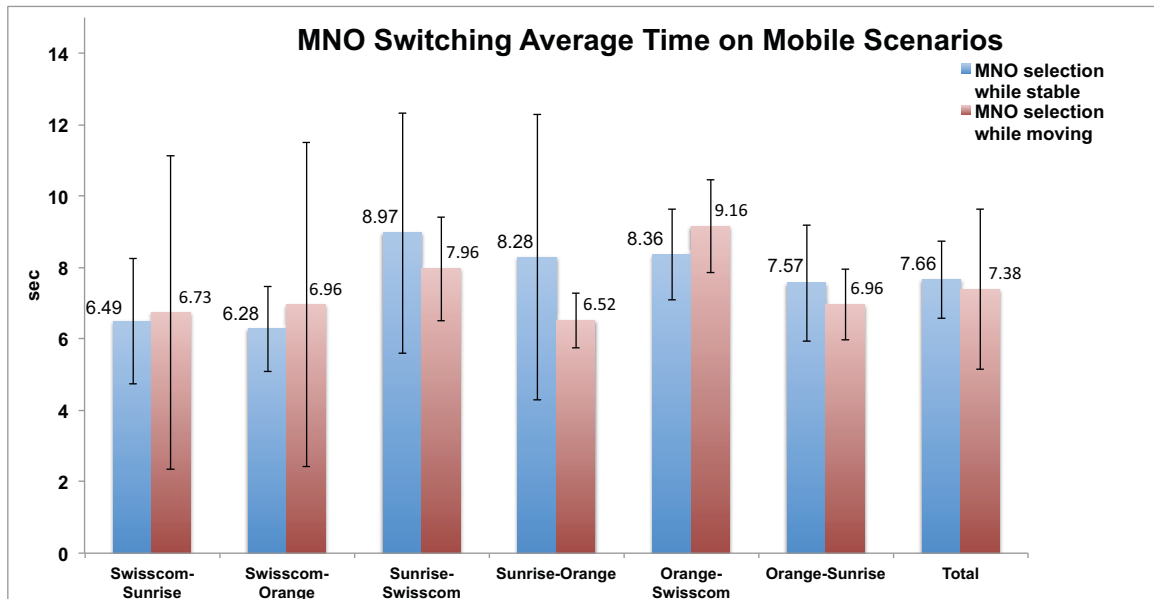


Figure 19: MNO Switching Average Time on Mobile Scenarios

bar corresponds to all switches performed from the indicated MNO to another. The error bars indicates the standard deviation. The first blue bar shows the average switching time from Swisscom to Sunrise at the same location. The first red bar in contrary shows the average switching time from Swisscom to Sunrise while moving. The first MNO in the caption below the bar is always the MNO where the device was registered first and the second MNO is the one that has been switched to. The large standard deviation results from big maximum values which can be seen on Figure 20. The implemented tests always tries to register to an MNO until it was successful. That is the reason why in the test case

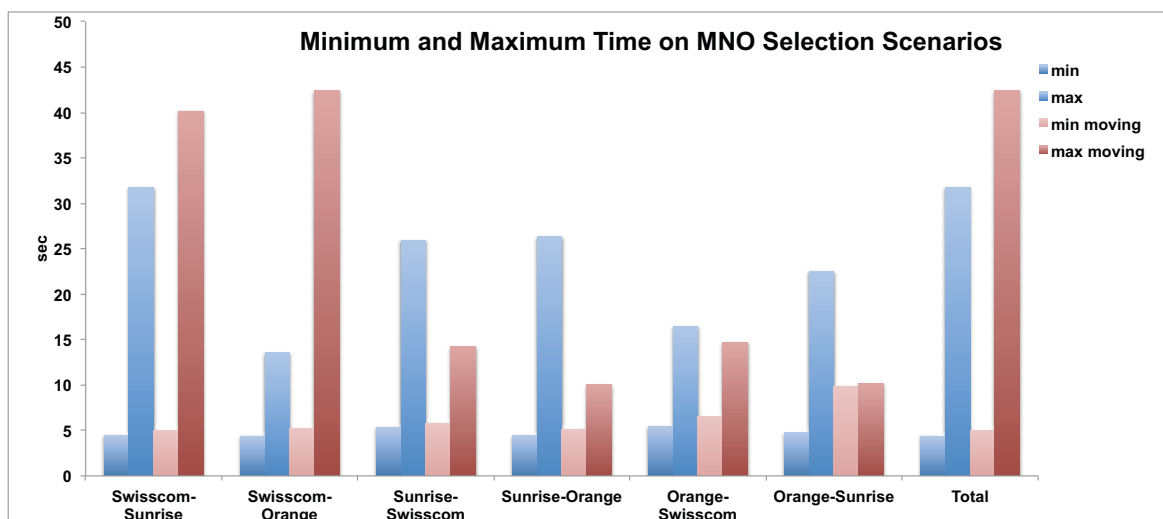


Figure 20: Min and Max Time on MNO Selection Scenarios

while moving the maximum values appear to be very high compared to the experiments at the same location.

Figure 21 shows that there is no significant difference in the average switching time if it is distinguished by the signal strength after a successful selection.

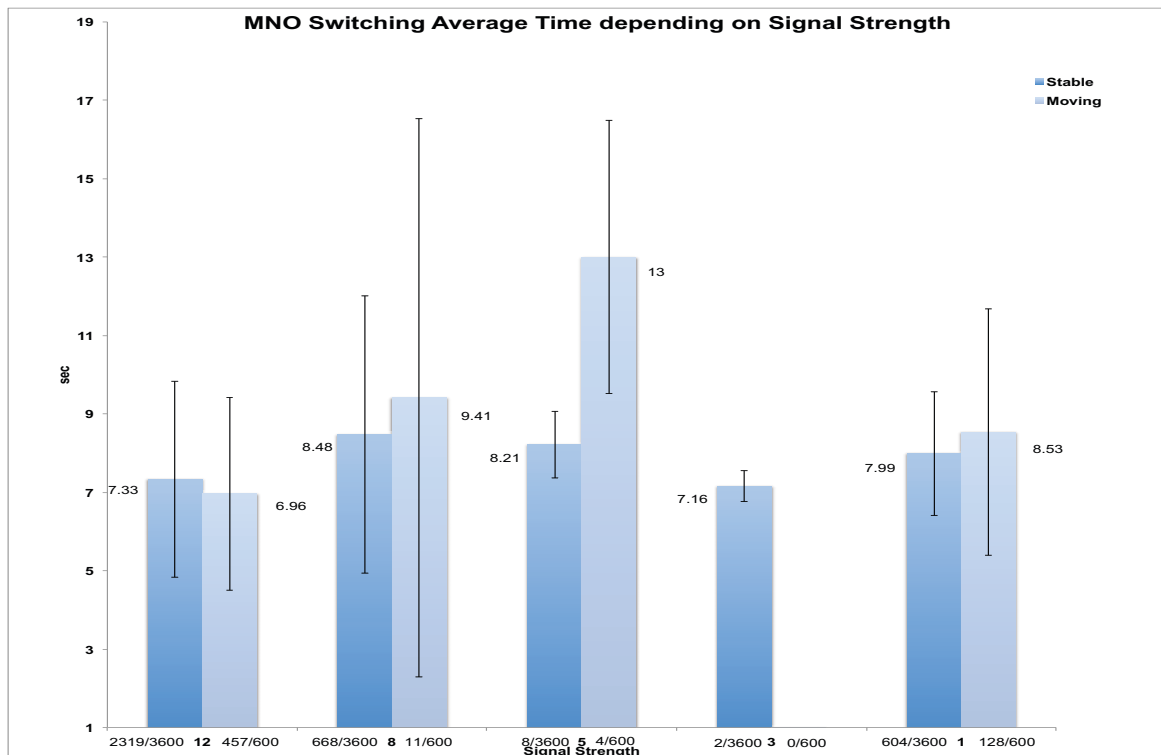


Figure 21: MNO Switching Average Time Signal Strength

Thus the average switching time is not dependent on the signal strength. The numbers below the bars in Figure 21 indicates the sum of how many time the signal strength occurred out of the total test hops. Figure 22 gives an overview of how the switching time depends on the time of day. The data for the bargraph is collected out of the total 3600 hops collected on a weekday. The error bars indicate the standard deviation of the measurements.

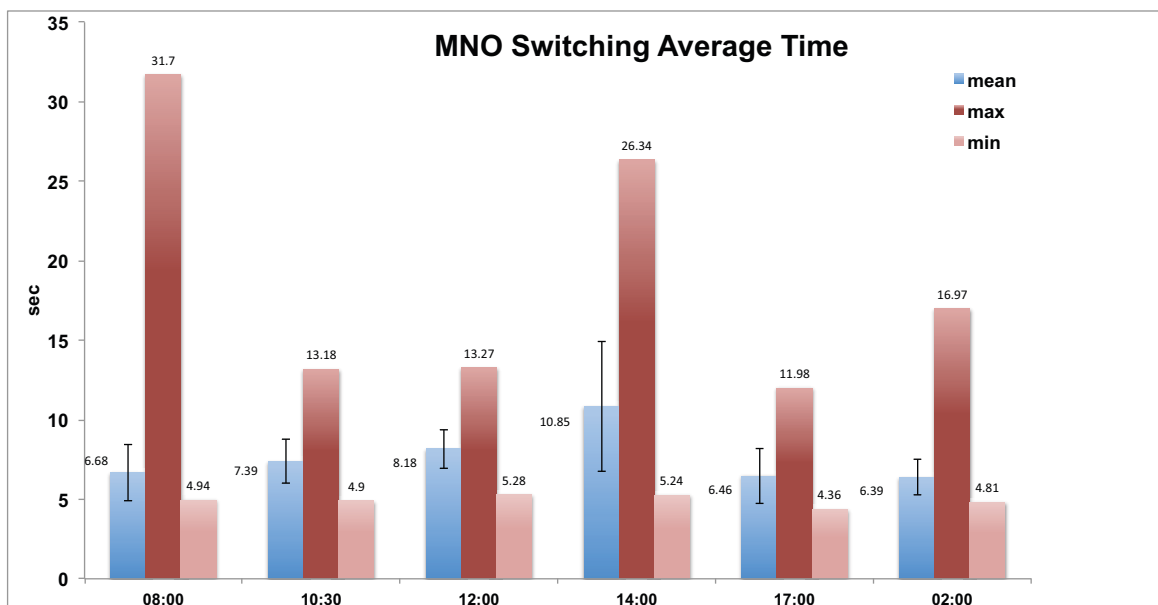


Figure 22: MNO Switching Average Time

The bars of Figure 23 illustrates the cases where the time to switch a MNO took more than ten seconds.

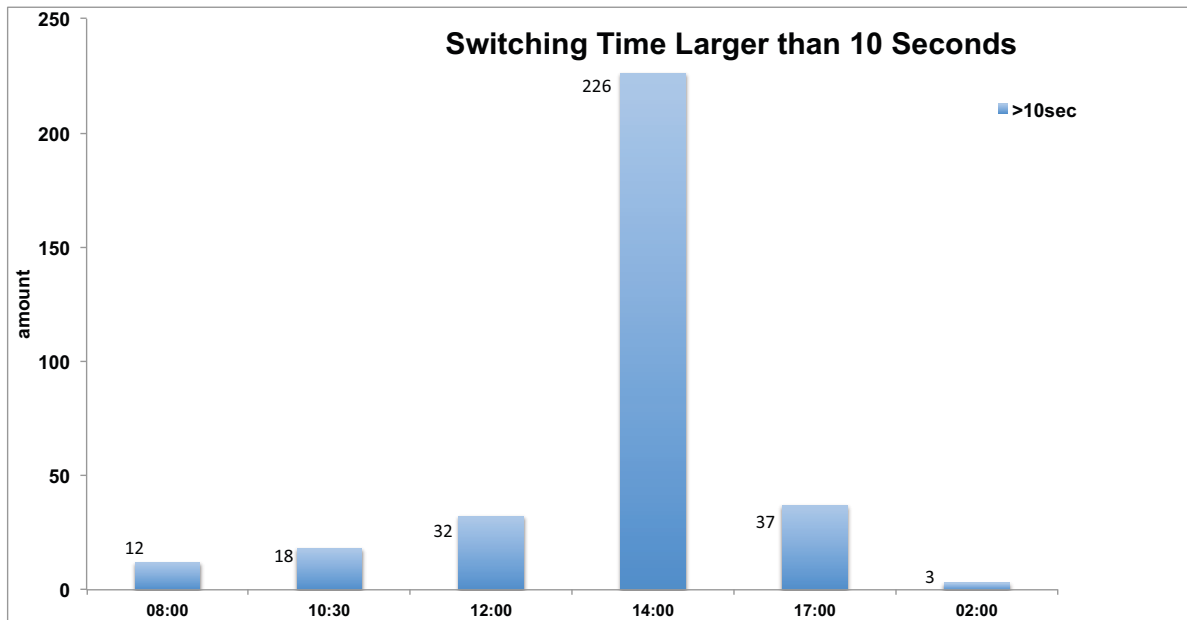


Figure 23: Large MNO Switching Time

In addition to the automated tests a manual test has been performed to get the dimension of how long it takes for a phone call in AbaCUS. To test this, a mock Au<sup>2</sup> server has been running in a local network. The time was taken after the dial button in the AbaCUSApp was pressed until the callee's phone was ringing. Because the Au<sup>2</sup> mock server simulates that Swisscom always wins, the test has been done ten times each when the callee's device had to switch from Orange to Swisscom and ten times when it had to switch from Sunrise to Swisscom. In other ten cases the time was measured when the MNO did not have to change because it already was Swisscom. The results are summarized in Figure 24.

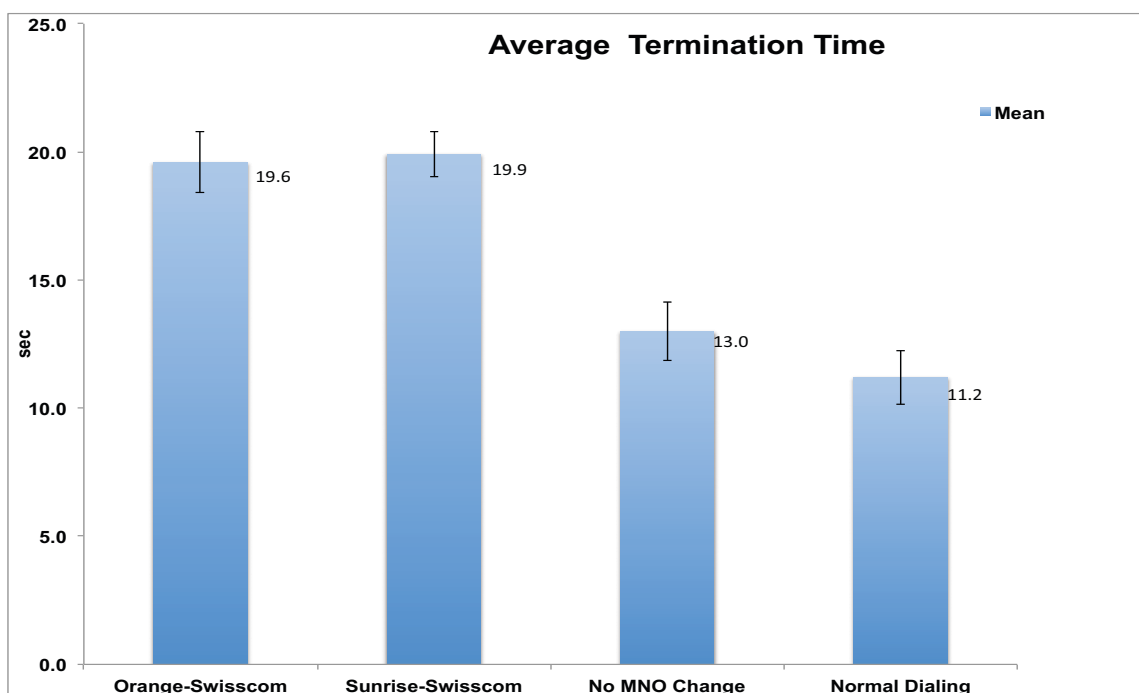


Figure 24: Average Termination Time

The different bars indicate the average time consumed for a call termination where the error bars are representing the standard deviation of the measurements. In AbaCUS, the average time of a call process is mainly dependent on the dialing time. This is apparent in Figure 24 where the average time in the case where no MNO change happened is still comparable with the normal dialing case where AbaCUS is not involved. The difference of this time to the cases where the MNO has been switched corresponds to the average MNO selection time that has been evaluated. The first MNO in caption below the bars indicates to which MNO the callee's device was registered before the call. The second MNO is the winning one which the callee's device had to switch to. The third bar means that the callee's device was already registered to the winning MNO. The last bar shows the average time when dialing with a normal dialer without AbaCUS.

Besides the evaluation of the MNO switching mechanism, the mechanism to get available networks has been evaluated with respect to time and power consumption. For that reason, available operators have been searched 100 times in a stable position and 300 times when moving from Zurich to Lucerne by train. Besides recording the used battery and consumed time the available networks have been logged. In the case when moving the evaluation returned an operator list in 292 times out of the 300. In seven cases when the scan failed the consumed time was only one to four hundredth of a second and thus negligible like the failed scans in the stable test. The bars in Figure 25 illustrate the average time needed for a successful scan of available operators where the error bars are indicating the standard deviation.

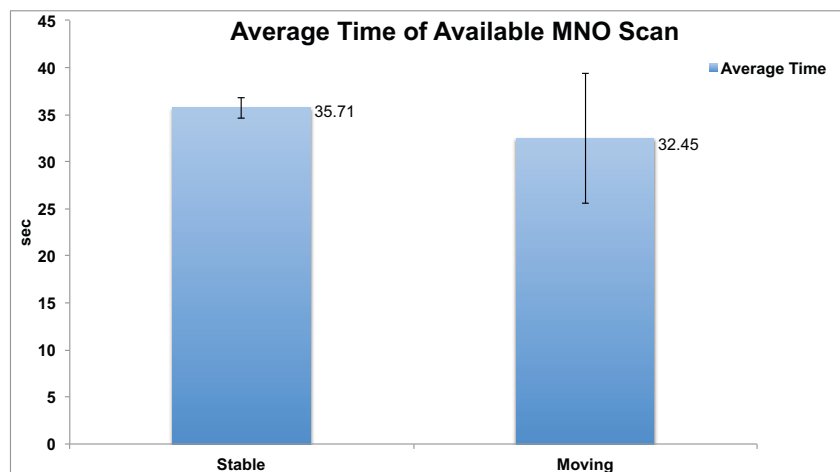


Figure 25: Average MNO Scan Time

## 4.2 Power Consumption

The power consumption is very critical in a mobile system. If the developed mechanism would consume the whole battery within a few network hops it would not be usable in practice. Hence an estimation of the power consumption has been made. To measure the power consumption, the battery level was determined before the test run and after the test has been performed according to [36]. The difference of these levels was the final battery consumption in percentage of the battery energy. The assumption was that the battery health is in a new ideal condition. This is appropriate because the device was only used for scientific purposes and therefore not heavily used. The assumption had to be made because there currently is not an application that measures the current battery energy or an application to measure the energy used per application. The total energy of a new SGS2 battery is 6.11 Wh according to the manufacturer.

In other words the total energy of a new battery is 21'996 J (Figure 26 (1)). During the test the display of the device remained turned off. In the test case where the location was stable the measured battery consumption was 14%. This corresponds to the energy consumption of 3079.44 J. To get the total power for the MNO switching mechanism the consumed energy has to be divided by the total experiment time. This calculation includes the energy needed for capturing the signal strength and the cell ID five times for each MNO switching measurement. However, the energy consumed on this process is relatively small compared to the energy demanded by the MNO switching process. This leads to a total of 0.5406 W consumed power for the MNO switching mechanism (Figure 26 (2)).

$$E = 6.11 \text{ Wh} \times 3600 \text{ s} = 21996 \text{ J} \quad (1)$$

$$P_{\text{tot}} = \frac{E(\text{J}) \times \text{battery}_{\text{used}}}{t_{\text{experiment}}(\text{s})} = \frac{21996 \text{ J} \times 0.14}{5696 \text{ s}} = 0.5406 \text{ W} \quad (2)$$

Figure 26: Power Consumption Calculation

This value seems high but feasible if it is compared to the power consumption of the talk mode in 3G networks (Table 2).

Table 2: SGS2 Battery [28]

Consumption Kind	2G	3G
Total Talk Time	1100 min	520 min
Calculated Talk Power	0.333 W	0.705 W
Stand-by Time	710	610
Calculated Stand-by Power	8.6 mW	10 mW

The solution process of the calculation can be seen in Appendix D. The same test has been performed while moving from Zurich to Lucerne by train. The test lasted 7404 seconds where 22% of the battery was consumed. This corresponds to a total power consumption of 0.6536 Watt. While comparing the values of both tests it is evident that in the case the mobile device is moving the power consumption is 20.9% higher than the power consumption when the position is stable (Table 3).

Table 3: MNO Switching Power Consumption

Kind	Power Consumption
Total Power Stable	0.5406 W
Total Power Moving	0.6536 W

Table 4 summarizes the power consumption of the available MNO scans.

The total power when stable was calculated from the 100 scans also used for the time consumption evaluation. This test took 3571 seconds. The total power when moving was calculated from the experiment with 300 scans when moving from Zurich to Lucerne. The elapsed time of this test was 9510 seconds.

Table 4: Available MNO List Power Consumption

Kind	Power Consumption
Total Power Stable	0.123 W
Total Power Moving	0.231 W

### 4.3 Bandwidth Consumption

The bandwidth consumption of a call process has been measured with an application called Bandwidth Monitor [14]. This application shows the amount of data sent and received per application. Table 5 summarizes the different messages that have been measured. The

Table 5: Bandwidth Consumption

Activity	Received (Bytes)	Sent (Bytes)
Register GCM	128	457
Request Service	358 (1)	796 (4)
Change MNO	256 (2)	276 (3)
Unregister GCM	128	437

registration process to the GCM server for the push notifications is necessary the first time the application starts as well as when the SIM card is changed. To deactivate this push notifications the application has to be unregistered from the GCM server. A service request is always sent from a caller to the Au<sup>2</sup> server. Figure 27 summarizes a call process in

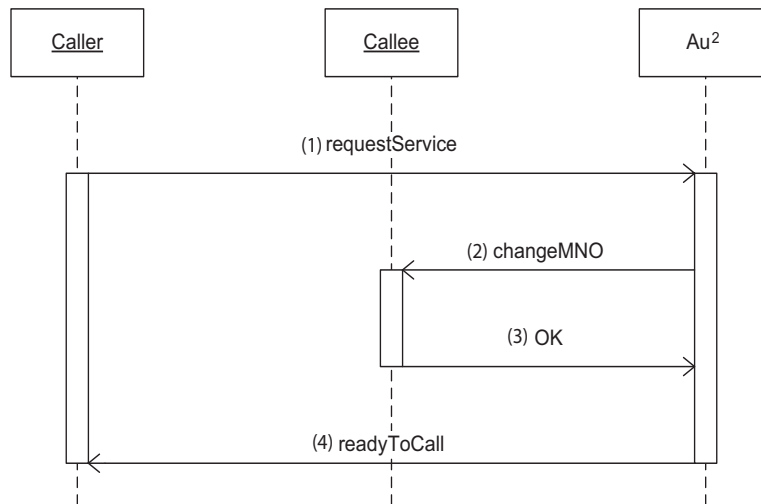


Figure 27: AbaCUS Call Process

AbaCUS. The structure of the messages from the caller to the Au<sup>2</sup> server is known, because it is sent as JSON [32]. However, in the other direction the structure is not known because it is handled by GCM. There, the needed attributes are committed as a key value pair where the key indicates the type of the attribute and the value the attribute itself. All AbaCUS messages with their attributes are shown in Appendix E. The numbers before the message names in Figure 27 corresponds to the numbers in brackets in Table 5.

---

## 5 Conclusions

The goal of this bachelor thesis was to implement an automatic, on-demand Mobile Network Operator (MNO) selection mechanism on Android devices. This goal has been accomplished by the implementation of an Android service to change the MNO programmatically. This mechanism has been evaluated according to its energy efficiency and time consumption. The evaluation has shown, that the mechanism would be practicable, because the time consumption was in an acceptable extent. The dialing took most of the time used in a AbaCUS call. The time consumption for the switching mechanism as well as the communication to the server was negligibly low. The evaluation showed that the power consumption of a single MNO switch is high, but in the same dimension as when a phone call is performed. In order to face the requirements of AbaCUS, an Android application has been implemented which makes use of the MNO selection mechanism. The application is able to receive the user's location and the available networks. To get an estimation of how much data the AbaCUS messages require, an evaluation of the bandwidth consumption has been done. The existing knowledge of how to use the internal Android API has been used to gain access to more methods for the connection with the GSM modem which the open Android API does not provide. However, it has to be kept in mind, that this workaround is not the best method, since the internal API is not listed and also may change in the future. If new models of network selection become accepted that require automatic, on-demand MNO selection, such mechanisms should be published in the open API.

The developed application is only a prototype. For a full working application all AbaCUS messages have to be specified, such that the application as well as the corresponding server can work together. For this prototype no human computer interaction principles have been considered. Thus some actions do not give an immediate user feedback. The prototype is specifically developed for a SGS2. On other devices the Graphical User Interface (GUI) might not be displayed in a proper way. For the developed mechanism no unit tests have been written. It has been assumed that the used methods to switch MNOs are already tested by Google. Last but not least, the AbaCUS application has been tested to ensure that the desired MNO switches were performed as expected.



---

## 6 Future Work

Due to the fact that until now no automatic, on-demand MNO selection mechanism has been published the findings of this thesis can be an instrument to help on research for new MNO selection algorithms. The implemented mechanism can for example be used to provide a proof of concept for a network switching selection model that minimize the non-ionizing radiation of Android devices during use [47]. Further research could include searching for mechanism for an automatic and on-demand MNO selection on other widespread platforms.

To demonstrate the technical feasibility of AbaCUS, the missing parts like the Au<sup>2</sup> server and the infrastructure to communicate with the devices should be implemented. The application that has been developed within this thesis could then be integrated in this system. Since the implemented mechanism has not considered any security issues, future work could address this topic as well. Due to the fact that the developed application is a proof of concept for Android based devices there are many things a productive application should consider. For example a solution to provide the developed mechanism on other smartphones is proposed. Other possible future work is the implementation of a stable and well tested multi-platform AbaCUS application where human computer interaction principles are also taken into account. For simplicity reasons user inputs are not checked for correctness so far. *E.g.*, the phone number that is dialed is not checked for the right pattern. It is assumed that the user always inserts correct data.

The development of a method to change the MNO programmatically is novel and has not been published before, so far as is known. Therefore, it opens the field for many different usages, such as the MNO selection based on various criteria/metrics.

---

## References

- [1] AbaCUS Project, URL:[www.abacusproject.eu](http://www.abacusproject.eu), Visited in August 2013.
- [2] Activity, URL:<http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle>, Visited in July 2013.
- [3] Android API Level, URL:<http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>, Visited in August 2013.
- [4] Android Architecture - The Key Concepts of Android OS, URL:<http://www.android-app-market.com/android-architecture.html>, Visited in July 2013.
- [5] Android Interface Definition Language (AIDL), URL:<http://developer.android.com/guide/components/aidl.html>, Visited in August 2013.
- [6] Android Source Code, URL:[http://greppcode.com/snapshot/repository.greppcode.com/java/ext/com.google.android/android/4.0.4\\_r2.1/](http://greppcode.com/snapshot/repository.greppcode.com/java/ext/com.google.android/android/4.0.4_r2.1/), Visited in July 2013.
- [7] Android, the world's most popular mobile platform, URL:<http://developer.android.com/about/index.html>, Visited in July 2013.
- [8] Android your (my) way, URL:<http://omri.org.il/2012/05/25/android-your-my-way/>, Visited in July 2013.
- [9] M. Anvaari and S. Jansen: *Evaluating architectural openness in mobile software platforms*; Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 85-92). ACM. August, 2010.
- [10] Application Fundamentals, URL:<http://developer.android.com/guide/components/fundamentals.html>, Visited in July 2013.
- [11] AT Commands, URL:<http://www.radioraiders.com/gsm-at-commands.html>, Visited in July 2013.
- [12] AT+COPS - Operator selection, URL:<http://www.activexperts.com/mmtoolkit/at/commands/?at=%2BCOPS>, Visited in July 2013.
- [13] Backsmali / Smali Manager, URL:<http://forum.xda-developers.com/showthread.php?t=2311766>, Visited in August 2013.
- [14] Bandwidth Monitor, URL:<https://play.google.com/store/apps/details?id=org.network&hl=de>, Visited in August 2013.
- [15] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. R. Sadeghi and B. Shastri: *Towards taming privilege-escalation attacks on Android*; Proceedings of the 19th Annual Symposium on Network and Distributed System Security, February, 2012.
- [16] Can a telephony.Phone object be instantiated through the sdk? ,URL:<http://stackoverflow.com/questions/2143754/can-a-telephony-phone-object-be-instantiated-through-the-sdk?lq=1>, Visited in July 2013.
- [17] N. Carlo: *Einblick in die Dalvik Virtual Machine*; IMVS Fokus Report, Vol. 3, No. 1, pp 5-12, 2009.
- [18] C. Cockings: *GSM AT Command Set*; UbiNetics Ltd Cambridge Technology Centre, Melbourn, 2001.
- [19] Custom ROMs For Android Explained - Here Is Why You Want Them, URL:<http://www.androidpolice.com/2010/05/01/custom-roms-for-android-explained-and-why-you-want-them/>. Visited in July 2013.

- 
- [20] CyanogenMod System Signature, URL:[https://github.com/CyanogenMod/android\\_build/tree/gingerbread/target/product/security](https://github.com/CyanogenMod/android_build/tree/gingerbread/target/product/security), Visited in July 2013.
  - [21] dex2jar, URL:<http://code.google.com/p/dex2jar/downloads/list>, Visited in August 2013.
  - [22] Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); AT command set for User Equipment (UE) (3GPP TS 27.007 version 8.5.0 Release 8), October, 2008.
  - [23] Downloading and Building, URL:<http://source.android.com/source/building.html>, Visited in August 2013.
  - [24] Eclipse IDE, URL:<http://www.eclipse.org>, Visited in August 2013.
  - [25] Extracting the Contents of a JAR File, URL:<http://docs.oracle.com/javase/tutorial/deployment/jar/unpack.html>, visited in July 2013.
  - [26] I. Forman, N. Forman and J. V. Ibm: *Java reflection in action*. 2004.
  - [27] Frogmobile, URL:<http://www.frogmobile.gr/en/default.aspx>, Visited in August 2013.
  - [28] Galaxy S II, URL:<http://www.samsung.com/uk/consumer/mobile-devices/smart-phones/android/GT-I9100LKAXEU-spec>, Visited in August 2013.
  - [29] Google Cloud Messaging for Android, URL:<http://developer.android.com/google/gcm/index.html>, Visited in July 2013.
  - [30] How to get available network operators? (using RIL and non-API methods), URL:<http://stackoverflow.com/questions/15929591/how-to-get-available-network-operators-using-ril-and-non-api-methods>, Visited in July 2013.
  - [31] How to talk to the Modem with AT commands, URL:<http://forum.xda-developers.com/showthread.php?t=1471241>, Visited in July 2013.
  - [32] JSON, URL:<http://www.json.org>, Visited on August 2013.
  - [33] JSON Tutorial, URL:<http://www.w3schools.com/json/default.asp>, Visited in August 2013.
  - [34] Manifest, URL:<http://developer.android.com/guide/topics/manifest/manifest-element.html>, Visited in July 2013.
  - [35] Z. Mednicks, L. Dornin, B. Meike and M. Nakamura: *Programming Android*; First Edition, O'Reilly Media Inc., Sebastopol, July, 2011.
  - [36] Monitoring the Battery Level and Charging State, URL:<http://developer.android.com/training/monitoring-device-state/battery-monitoring.html>, Visited in August 2013.
  - [37] Notepad++, URL:<http://notepad-plus-plus.org>, Visited in August 2013.
  - [38] Orange, URL:<http://www1.orange.ch>, Visited in August 2013.
  - [39] Planetsim, URL:[http://planetsim.gr/home\\_page/](http://planetsim.gr/home_page/), Visited in August 2013.
  - [40] Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2012 und 2016, URL:<http://de.statista.com/statistik/daten/studie/182363/umfrage/prognostizierte-marktanteile-bei-smartphone-betriebssystemen/>, Visited in July 2013.
  - [41] Programatically connecting to another Network operators, URL:<http://stackoverflow.com/questions/2373727/programatically-connecting-to-another-network-operators>, Visited in July 2013.
-

- 
- [42] Programatically setting network mode, URL:<http://stackoverflow.com/questions/16149809/programatically-setting-network-mode>, Visited in July 2013.
  - [43] ResultReceiver, URL:<http://developer.android.com/reference/android/os/ResultReceiver.html>, Visited in August 2013.
  - [44] J. Schiller: *Mobile Communications*; Second Edition, Pearson Education, Harlow, 2003.
  - [45] J. Schiller: *Solution Manual for Mobile Communications*; Second Edition, Berlin.
  - [46] D. Seal: *ARM architecture reference manual*. Pearson Education, 2000.
  - [47] J. M. Seigneur, X. Titi, T. Maliki: *Towards Mobile/Wearable Device Electrosmog Reduction through Careful Network Selection*; Proceedings of the 1st Augmented Human International Conference. ACM, April, 2010.
  - [48] Send AT commands to USB modem, URL:<http://brunomgalmeida.wordpress.com/2012/04/06/send-at-commands-to-usb-modem/>, Visited in July 2013.
  - [49] Service, URL:[http://developer.android.com/reference/android/app/Service.html#START\\_STICKY](http://developer.android.com/reference/android/app/Service.html#START_STICKY), Visited on August 2013.
  - [50] Setting Network Operator, URL:<http://www.basic4ppc.com/android/forum/threads/setting-network-operator.24942/>, Visited in July 2013.
  - [51] M. Shen and J. Jiang: *Design and implementation of Radio Interface layer in Android video telephone system*; International Conference on Computer Science and Network Technology (ICCSNT), 2011 , Vol. 3, pp 1429 - 1432, December, 2011.
  - [52] SlickEdit, URL:<http://www.slickedit.com>, Visited in August 2013.
  - [53] smali/baksmali, URL:<https://code.google.com/p/smali/>, Visited in August 2013.
  - [54] SQLite, URL:<http://www.sqlite.org>, Visited in August 2013.
  - [55] Stackoverflow, URL:<http://stackoverflow.com>, Visited in August 2013.
  - [56] Sunrise, URL:<http://www1.sunrise.ch>, Visited in August 2013.
  - [57] Swisscom, URL:<http://www.swisscom.ch>, Visited in August 2013.
  - [58] Telephony Manager, URL:<http://developer.android.com/reference/android/telephony/TelephonyManager.html>, Visited in July 2013
  - [59] The Security Manager, URL:<http://docs.oracle.com/javase/tutorial/essential/environment/security.html>, Visited in July 2013.
  - [60] Trail: The Reflection API ,URL:<http://docs.oracle.com/javase/tutorial/reflect/>, Visited in July 2013.
  - [61] C. Tsiaras and B. Stiller: *Challenging the Monopoly of Mobile Termination Charges with an Auction-based Charging and User-centric System (AbaCUS)*; Networked Systems (NetSys), March, 2013.
  - [62] Using internal (com.android.internal) and hidden (@hide) APIs [Part 1, Introduction], URL:<https://devmaze.wordpress.com/2011/01/18/using-com-android-internal-part-1-introduction/>, Visited in July 2013.
  - [63] T. Vaattovaara: *Analysis of Modem Integration in Open Source Smartphone Platforms*; Master Thesis, Oulu University of Applied Sciences, March, 2011.
  - [64] XDA Developers, URL:<http://www.xda-developers.com>, Visited in August 2013.

- 
- [65] 20% iOS - 60% Android: Umfangreiche Studie zur Smartphone-Nutzung in Deutschland, URL:<http://www.iphone-ticker.de/20-ios-60-android-umfangreiche-studie-zur-smartphone-nutzung-in-deutschland-44098/>, Visited in July 2013.
- [66] 3rd Generation Partnership Project; Technical Specification Group Terminals; AT command set for User Equipment (UE), Release 6, June, 2003.

---

## Appendix A: Abbreviations

AbaCUS	Auction-based Charging an User-centric System
ADT	Android Developer Tools
Au <sup>2</sup>	Auction Authority
API	Application Programming Interface
CCS	Cloud Connection Server
DVM	Dalvik Virtual Machine
GCM	Google Cloud Messaging
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IDE	Integrated Developer Environment
IMEI	International Mobile Equipment Identity
IPC	Interprocess Communication
MCC	Mobile Country Code
MNC	Mobile Network Code
MNO	Mobile Network Operator
MSISDN	Mobile Subscriber Integrated Services Digital Network Number
OS	Operating System
RIL	Radio Interface Layer
SGS2	Samsung Galaxy S II
UUID	Universally Unique Identifier
VM	Virtual Machine
XML	Extensible Markup Language

---

## Appendix B: Glossary

AT command	Specific command language to access modems.
CCS	The GCM Cloud Connection Server allows third party servers to communicate with Android devices.
GCM	A service to send push notifications with payload between an Android device and a server.
RIL	Middle layer between Android applications and wireless module that sends AT commands to the baseband.

---

## Appendix C: Tools and Environments

Table 6: Tools and Frameworks Used

Tool	Description	Version	Source
ADT (Android Development Tools)	Plug-in for Eclipse, which provides a development environment for building Android applications	21.1.0	<a href="http://developer.android.com/tools/sdk/eclipse-adt.html">http://developer.android.com/tools/sdk/eclipse-adt.html</a>
baksmali	Disassembler for the dex format used by dalvik	1.4.2	<a href="https://code.google.com/p/smali/downloads/detail?name=baksmali-1.4.2.jar&amp;can=2&amp;q=">https://code.google.com/p/smali/downloads/detail?name=baksmali-1.4.2.jar&amp;can=2&amp;q=</a>
Bandwidth Monitor	Android Application This to keep a track of bandwidth usage by application.	1.0.6	<a href="https://play.google.com/store/apps/details?id=org.network&amp;hl=de">https://play.google.com/store/apps/details?id=org.network&amp;hl=de</a>
dex2jar	Converter for .dex to .jar file format	0.0.9.15	<a href="http://code.google.com/p/dex2jar/downloads/list">http://code.google.com/p/dex2jar/downloads/list</a>
Eclipse	Integrated development environment (IDE)	Juno Service Release 2	<a href="http://www.eclipse.org">http://www.eclipse.org</a>
Google Cloud Messaging for Android Library		3	
Google Play services		7	
jarsigner	Tool to verify the signatures and integrity of signed JAR files		<a href="http://docs.oracle.com/javase/7/docs/tech-notes/tools/windows/jarsigner.html">http://docs.oracle.com/javase/7/docs/tech-notes/tools/windows/jarsigner.html</a>
Notepad++	Source Code Editor	6.4.2	<a href="http://notepad-plus-plus.org">http://notepad-plus-plus.org</a>



---

Tool	Description	Version	Source
puTTY	Open-source terminal emulator, serial console and network file transfer application	0.62	<a href="http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html">http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html</a>
signapk	Program to sign an application with a system signature	0.3.1	<a href="http://code.google.com/p/signapk/downloads/list">http://code.google.com/p/signapk/downloads/list</a>
SlickEdit	Cross-platform, multi-language code editor	18.0.0.13	<a href="http://www.slickedit.com">http://www.slickedit.com</a>
smali	Assembler for the dex format used by dalvik	1.4.2	<a href="https://code.google.com/p/smali/downloads/detail?name=smali-1.4.2.jar">https://code.google.com/p/smali/downloads/detail?name=smali-1.4.2.jar</a>

---

## Appendix D: Power Consumption Calculation

$$E = 6.11\text{Wh} \times 3600\text{s} = 21996\text{J}$$

$$P_{\text{talkTime2G}} = \frac{E(\text{J})}{t_{\text{talkTime2G}}(\text{s})} = \frac{21996}{1100\text{min} \times 60\text{s}} = 0.333\text{W}$$

$$P_{\text{talkTime3G}} = \frac{E(\text{J})}{t_{\text{talkTime3G}}(\text{s})} = \frac{21996}{520\text{min} \times 60\text{s}} = 0.705\text{W}$$

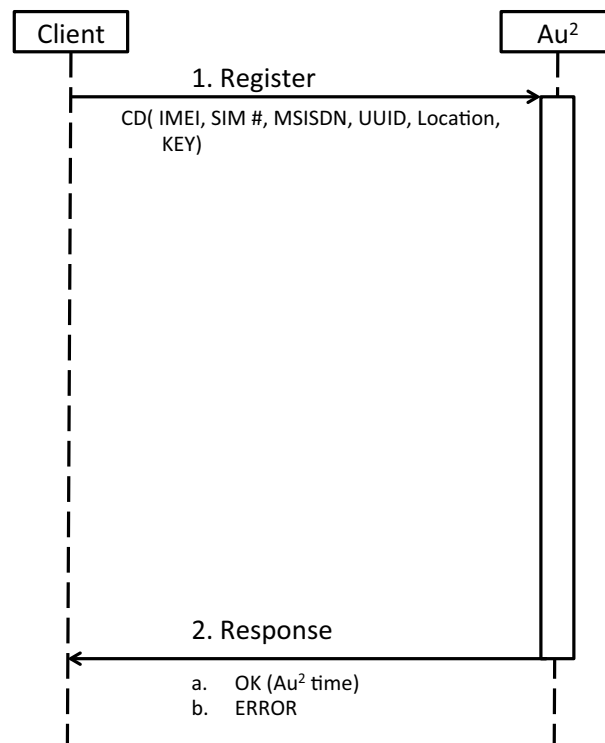
$$P_{\text{standbyTime2G}} = \frac{E(\text{J})}{t_{\text{StandbyTime2G}}(\text{s})} = \frac{21996}{710\text{h} \times 3600\text{s}} \times 10^3 = 8.6\text{mW}$$

$$P_{\text{standbyTime3G}} = \frac{E(\text{J})}{t_{\text{StandbyTime3G}}(\text{s})} = \frac{21996}{610\text{h} \times 3600(\text{s})} \times 10^3 = 10\text{mW}$$

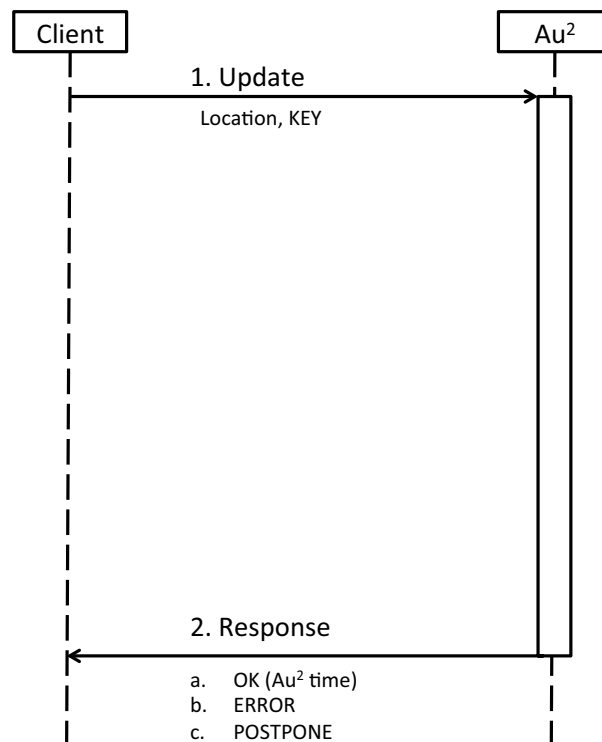
---

## Appendix E: AbaCUS Messages

The registration takes place every time that a client connects.



The update takes place periodically. The time interval should be configurable.

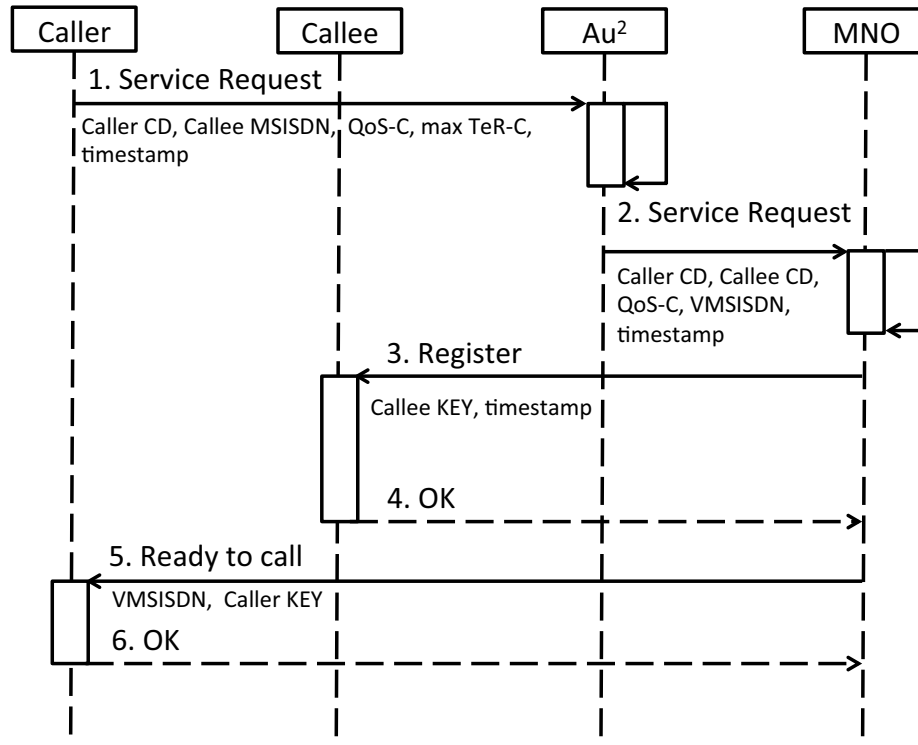


This is the case that the callee is not busy.

For every step there is a time out. For example from step 1 on, the caller has to wait for a period of time before step 5. If step 5 takes too long the caller will assume that the call is not possible to be placed. This time frame should be configurable in the client settings.

In step 2 if the time that the Service Request has been received is not close enough with the timestamp the step 3 will not follow (this time frame should also be configurable).

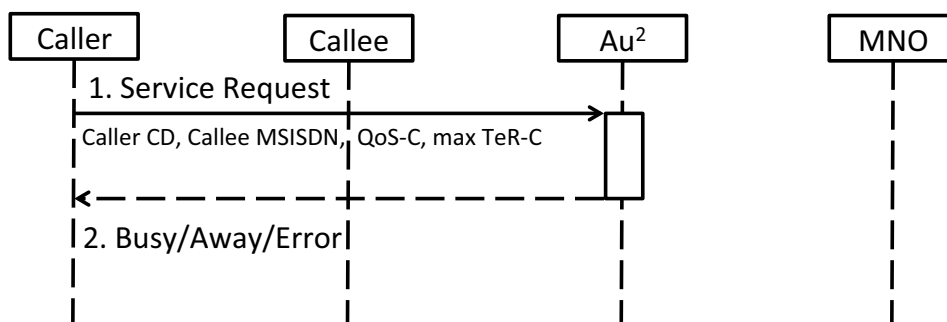
The same principle with the time stamp is followed by the callee in step 3 as well.



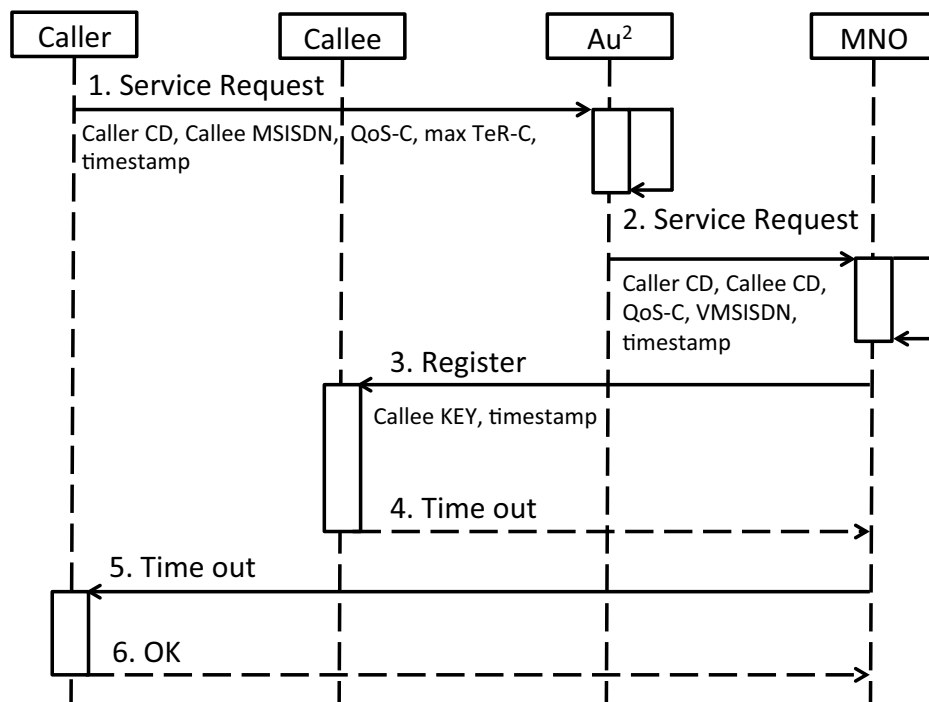
In this case the callee might be already on a call (Busy)

Not registered in the Abacus (Away)

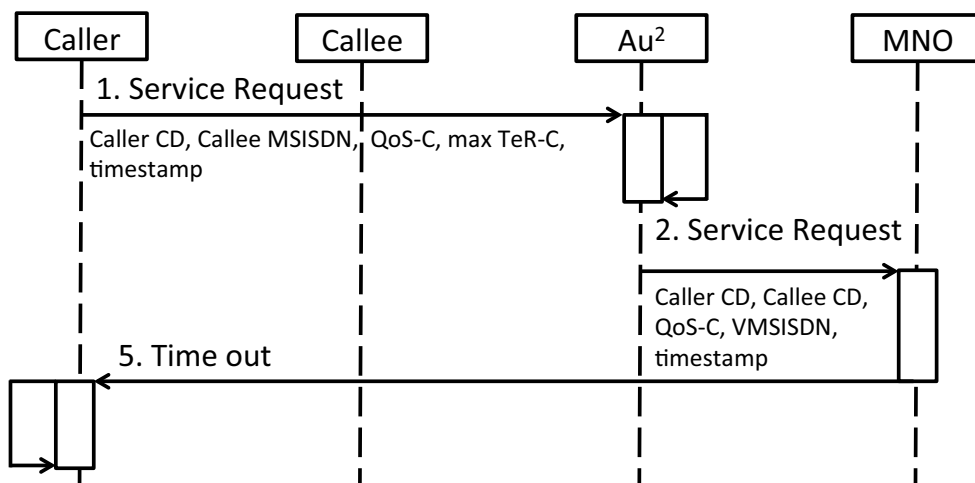
Or something else might be wrong (Error)



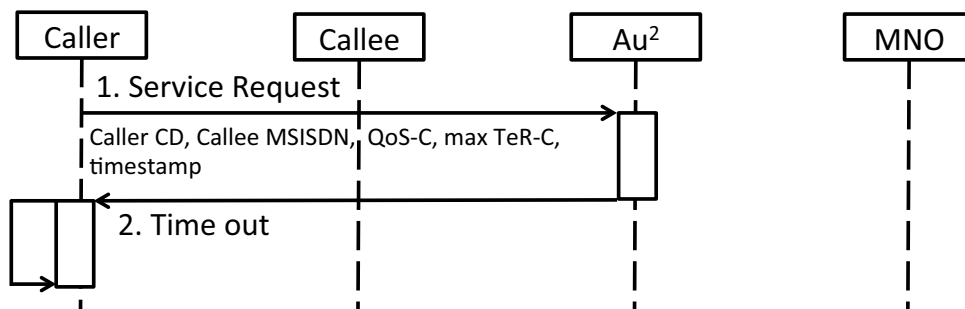
In this case the callee realizes that there is a time out so it does not register to the MNO.



In this case the MNO realizes that there is a time out, so it notifies the Caller that the call will not be completed



In this case the MNO realizes that there is a time out so notifies the Caller that the call will not be completed



---

## Appendix F: Contents of the CD-ROM

**AbaCUS Application** Project folder of the implemented Android application prototype including all source files and referenced libraries.

**Mock Au<sup>2</sup> Server** Project folder of the mocked server to demonstrate a call in AbaCUS.

**Bachelorthesis.pdf** Bachelor Thesis in PDF form.

**Abstract.txt** The abstract in English.

**Zusfsg.txt** The abstract in German.

**Evaluation** Folder which contains the source code of the mechanism evaluation as well as the evaluation results.

**Intermediate presentation.pdf** A pdf file of the Bachelor Thesis intermediate presentation.

**Related Work Papers** Folder which contains related work papers.