



NTNU – Trondheim
Norwegian University of
Science and Technology

End-User service composition framework and application

Rune Bleken Kulstad

Master of Science in Communication Technology

Submission date: June 2012

Supervisor: Mazen Malek Shiaa, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Name of Student: Rune Bleken Kulstad

The Master thesis is about developing a tool for handling value-added telecom services on Android devices. This application will allow end-users to compose their own services from existing service components. The Master thesis is a continuation of my previous project assignment. The assignment is also a continuation of previous project and master thesis assignments, where a platform for end-user service composition has been developed. The task is to further develop the end-user composition tool in terms of usability, usefulness and user-friendliness. The existing tool consists of two applications. The main task of this project is to make a new application that combines the functionality of the two existing applications, improves the user-friendliness and add new functionality. The assignment includes the following tasks:

- Propose improvements to the interaction mechanisms between end-users and the existing applications and implement these proposed improvements. This will result in a new user interface being developed and implemented.
- Develop and implement new functionality to the tool in the new application.
- Further develop a location-based service being developed in my previous project and integrate this in the new application.
- Integrate and implement new functionality and improvements proposed in my previous project to the new application.
- Work out new scenarios and develop, execute and test them in real execution environments based on the new application.
- Study and implement a solution to integrate with service composition framework aimed for service providers (on behalf of end-users)

The improvements that will be proposed will be based on end-user feedback from an on-going end-user evaluation process. The goal is that the new application will replace the two existing applications and that tool will be available to the public market by the end of the assignment period. This process is being performed within the scope of UbiCompForAll (a Norwegian research project in cooperation with Sintef).

Assignment given: End-User service composition framework and application

Supervisor: Mazen Malek Shiaa

Abstract

In today's public market Mobile phones has become a part of everyday life. The introduction of Smart Phones has created a new market for services and applications for the Smart Phones. Many of these users would benefit of customizing their own services to fulfil their needs. This can be achieved with end-user service composition. End-user service composition enables the user to compose their own services from already existing components to provide value added services. In this Master Thesis a service composition tool consisting of the two applications Easy Composer and EasyDroid is presented. The idea of the tool is that ordinary people without technical background will be able to quickly compose their own services in a simple manner. The existing tool has been in development for a while, but still lack some sufficient quality in terms of usability and utility for ordinary people to make use of it. Utility means what the tool can be utilized to, and usability means the user-friendliness and usefulness. In this Master Thesis a new system has been made for the service composition tool. The Easy Composer application has been discarded and new web based GUI has replaced its functionality. In addition, the EasyDroid application has been remade and a new server side has been developed. Furthermore, the communication between the different parts has been improved. The usability and utility of the previous system has been considerably improved in the new system. In other words the existing functionality has been made more user friendly and new functionality has been added to the tool. The goal is that the service composition tool would have the sufficient quality and novelty for ordinary users to embrace it.

Sammendrag

I dag har mobiltelefoner blitt en del av hverdagen til majoriteten av befolkningen. Innføringen av smart telefoner har skapt et nytt marked for tjenester og applikasjoner. Mange mobilbrukere vil ha nytte av å tilpasse sine egne tjenester for å oppfylle deres behov. Dette kan oppnås med tjeneste komposisjon for sluttbrukere. Tjeneste komposisjon gjør det mulig for brukeren å komponere sine egne tjenester fra allerede eksisterende komponenter. I denne masteroppgaven er et eksisterende tjeneste komponerings verktøy presentert. Dette verktøyet består av de to programmene Easy Composer og EasyDroid. Målet til verktøyet er at vanlige folk uten teknisk bakgrunn vil være i stand til raskt å komponere egne tjenester på en enkel måte. Den eksisterende verktøyet har vært i utvikling en stund, men mangler fortsatt tilstrekkelig kvalitet når det gjelder brukervennlighet og nytte for vanlige folk å gjøre bruk av det. I denne masteroppgaven har et nytt system blitt utviklet for tjeneste komposisjon verktøyet. Easy Composer applikasjonen har blitt forkastet og nytt nettbasert GUI har erstattet dets funksjonalitet. I tillegg har EasyDroid applikasjonen blitt gjenskapt og en ny server side implementert. Videre har kommunikasjonen mellom disse delene blitt forbedret. Brukervennligheten og nytten av verktøyet har blitt betydelig forbedret i det nye systemet. Med andre ord har eksisterende funksjonalitet blitt mer brukervennlig og ny funksjonalitet er lagt til verktøyet. Målet er at tjenesten komposisjons verktøyet ville ha tilstrekkelig kvalitet og nytteverdi for at vanlige personer vil bruke det.

Preface

This Master Thesis is written at the Department of Telematics in the Norwegian University of Science and Technology (NTNU) during the spring semester, 2012. The theme of this project work was defined by UbiCompForAll- Ubiquitous service Composition for All users. The project involves SINTEF, NTNU, Gintel, Tellu, and Wireless Trondheim. The Master Thesis is focused on the further development of the service composition tool applications, the Easy Composer application and EasyDroid. The Easy Composer application was created by Jens Einar Kielland Vaskinn and the EasyDroid application was created by John Edvard Reiten. The tool was further developed by Frank Mbaabu in his master thesis, adding Calendar and Map functionality to the Easy Composer and components to the EasyDroid. Frank Mbaabu and myself further developed the service composition tool in my project thesis, autumn 2011. This Master Thesis is a continuation of my previous project thesis in the autumn semester, 2011. In the Master Thesis period a new system was developed for the service composition tool.

I would like to thank my supervisor, Mazen Shiaa for giving me guidance and good advices during the Master Thesis period. I would also like to thank Frank K. Mbaabu for the cooperation and collaboration on developing the new system.

Rune Bleken Kulstad

June 7, 2012

Abbreviations

AJAX	Asynchronous JavaScript And XML
DOM	Document Object Model
GPS	Global Positioning System
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
JSNI	JavaScript Native Interface
RPC	Remote Procedure Call
SDL	Specification and Description Language
SOC	Service Oriented Computing
UML	Unified Modelling Language
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

List of Figures

FIGURE 1-1: CONTROL FLOW-BASED COMPOSITION (REPLICA OF THE FIGURE IN [5])	2
FIGURE 1-2: THE DEVELOPMENT PROCESS DURING THE MASTER THESIS PERIOD	3
FIGURE 2-1: USER-CENTERED DESIGN PROCESS	8
FIGURE 2-2: SDLVALID STRUCTURE OVERVIEW	15
FIGURE 2-3: GWT RCP	18
FIGURE 3-1: COMPOSITION AREA EASY COMPOSER	20
FIGURE 3-2: CALENDAR, EASY COMPOSER	22
FIGURE 3-3: MAP, EASY COMPOSER	23
FIGURE 3-4: MAP, EASY COMPOSER	24
FIGURE 3-5: EASYDROID ARCHITECTURE	25
FIGURE 3-6: EASYDROID GUI	26
FIGURE 3-7: BUDDY LIST SYSTEM OVERVIEW	27
FIGURE 3-8: USER INTERFACE, BUDDY LIST CLIENT	27
FIGURE 3-9: USER INTERFACE, BUDDY LIST AND MAP	29
FIGURE 4-1: SYSTEM OVERVIEW	31
FIGURE 4-2: LOG IN	34
FIGURE 4-3: PROFILES, LARGE SIZE	35
FIGURE 4-4: PROFILES, MEDIUM SIZE	37
FIGURE 4-5: PROFILES, SMALL SIZE	38
FIGURE 4-6: MAPS WITH LOCATION MARKER	40
FIGURE 4-7: MAPS WITH LOCATION TABLE	42
FIGURE 4-8: MAP, LOCATION SETTINGS WINDOW	43
FIGURE 4-9: MAP, OVERLAYS	44
FIGURE 4-10: MAP, LOCATION SHARING	45
FIGURE 4-11: CALENDAR INTERFACE, ADDING A TIME EVENT	47
FIGURE 4-12: CALENDAR, COMPACT GUI	48
FIGURE 4-13: GROUPS GUI, ADDING A MEMBER	50
FIGURE 4-14: GROUPS GUI, MEMBER SETTINGS	51
FIGURE 4-15: CLASS DIAGRAM, GWT GUI OVERVIEW	53
FIGURE 4-16: CLASS DIAGRAM, START UP FUNCTIONALITY	55
FIGURE 4-17: CLASS DIAGRAM, PROFILES FUNCTIONALITY	57
FIGURE 4-18: CLASS DIAGRAM, MAP FUNCTIONALITY	60
FIGURE 4-19: CLASS DIAGRAM, CALENDAR FUNCTIONALITY	63
FIGURE 4-20: CLASS DIAGRAM, GROUPS FUNCTIONALITY	65
FIGURE 4-21: CLASS DIAGRAM, EASYDROID OVERVIEW	67
FIGURE 4-22: CLASS DIAGRAM, EASYDROID COMPOSITION	69
FIGURE 4-23: CLASS DIAGRAM, EASYDROID ENGINE	71

FIGURE 4-24: SEQUENCE DIAGRAM, EASYDROID ENGINE	74
FIGURE 4-25: SEQUENCE DIAGRAM, FAST & MEMBER ACTIVATION.....	75
FIGURE 4-26: CLASS DIAGRAM, GWT SERVER SIDE.....	77
FIGURE 4-27: ER DIAGRAM, DATABASE.....	79
FIGURE 4-28: SEQUENCE DIAGRAM, SYSTEM COMMUNICATION	83
FIGURE 4-29: FAST ACTIVATION, SEQUENCE DIAGRAM	85
FIGURE 4-30: THE BUDDY LIST SYSTEM.....	88
FIGURE 4-31: THE BUDDY LIST, SEQUENCE DIAGRAM	89
FIGURE 4-32: MEMBER ACTIVATION, SEQUENCE DIAGRAM.....	89
FIGURE 4-33: CONFLICT RESOLUTION ACTIVITY DIAGRAM	93
FIGURE 4-34: COMPOSITION CONFLICT RESOLUTION ACTIVITY DIAGRAM	94
FIGURE 4-35: COMPONENT COMPATIBILITY MATRIX	96
TABLE 5-1: COGNITIVE WALKTHROUGH TASK DESCRIPTION.....	101
TABLE 5-2: QUESTIONNAIRE, TEST RESULTS.....	103
FIGURE 5-3: COLUMN DIAGRAM, QUESTIONNAIRE AVERAGE SCORE	104

Table of Contents

1. INTRODUCTION	1
1.1 MOTIVATION AND BACKGROUND	1
1.2 PROBLEM STATEMENT AND OBJECTIVES.....	2
1.3 STRUCTURE OF THE REPORT	5
2. THEORETICAL BACKGROUND AND RELATED WORK	7
2.1 USER-FRIENDLY DESIGN	7
2.1.1 <i>User-Centered Design</i>	7
2.1.2 <i>Usability Requirements & Measures</i>	9
2.1.3 <i>Design Guidelines & Principles</i>	10
2.1.4 <i>Paper Prototyping</i>	13
2.2 LOCATION BASED SERVICES	14
2.2.1 <i>Location-sensing</i>	14
2.2.2 <i>Google Latitude</i>	14
2.3 CONFLICT RESOLUTION	15
2.4 GOOGLE WEB TOOLKIT	17
2.4.1 <i>Communicating with server</i>	17
2.4.2 <i>GWT JSNI</i>	18
3. PREVIOUS SYSTEM	19
3.1 THE EASY COMPOSER	19
3.1.1 <i>The Composition area</i>	20
3.1.2 <i>The Calendar</i>	22
3.1.3 <i>The Map</i>	23
3.1.4 <i>The Groups</i>	24
3.2 THE EASYDROID	25
3.3 THE BUDDY LIST	27
4. SYSTEM OVERVIEW, DESIGN AND IMPLEMENTATION	31
4.1 SYSTEM OVERVIEW	31
4.2 CLIENT SIDE.....	33
4.2.1 <i>Graphical User Interface (GUI)</i>	33
4.2.1.1 <i>The Log In</i>	34
4.2.1.2 <i>The Profiles</i>	35
4.2.1.3 <i>The Map</i>	40
4.2.1.4 <i>The Calendar</i>	47
4.2.1.5 <i>The Groups</i>	50
4.2.2 <i>Architecture & Implementation of the GUI</i>	53
4.2.2.1 <i>The Start-Up Functionality</i>	55
4.2.2.2 <i>The Profiles Functionality</i>	57
4.2.2.3 <i>The Map Functionality</i>	60
4.2.2.4 <i>The Calendar Functionality</i>	63
4.2.2.5 <i>The Groups Functionality</i>	65
4.2.3 <i>The EasyDroid Engine</i>	67

4.2.3.1 The Start-Up Functionality	68
4.2.3.2 The Composition Functionality	69
4.2.3.3 The Engine Functionality.....	71
4.3 SERVER SIDE.....	77
4.4 USAGE, COMMUNICATION & INTERACTION	82
4.4.1 <i>Communication & Usage</i>	82
4.4.2 <i>Fast Activation</i>	85
4.4.3 <i>The Buddy List & Member Activation</i>	88
4.4.4 <i>The Conflict Resolution</i>	93
5. SYSTEM TEST & RESULTS	99
5.1 SYSTEM TESTING	99
5.2 USABILITY TESTING	102
5.2.1 <i>Field test</i>	102
5.2.2 <i>Focus Group</i>	106
6. DISCUSSION & FUTURE WORK	107
6.1 LIMITATIONS & IMPROVEMENTS	107
6.2 FUTURE DEVELOPMENT	110
7. CONCLUSION	111
APPENDIX.....	113
APPENDIX A: REFERENCES	113
APPENDIX B: QUESTIONNAIRE.....	115
APPENDIX C: XML REPRESENTATIONS.....	118
C1. <i>Exported Composition XML</i>	118
C2. <i>Fast Activation XML example</i>	120
C2. <i>Member Activation XML example</i>	120

1. Introduction

In the Master thesis period a new system for handling value-added telecom services on Android devices has been developed. The project is a continuation of my previous project (autumn 2011), and other previous projects and master thesis assignments where a tool for end-user service composition has been developed. The service composition tool consists of the two applications Easy Composer (Java FX application) and the EasyDroid (Android application).

1.1 Motivation and Background

Service composition is the creation of value added services by combining already existing services into a new service for a task or objective. In recent times mobile technology has evolved rapidly. The Mobile phones have become more powerful and wireless communication has become faster. With the introduction of Smart Phones the number of services has taken a giant leap. Today, there are numerous of services available, including telephony services, Internet services, personal management services and location-based services (LBS). The Service Composition can make use of these services by combining them in different ways in order to achieve a task. The goal with this project is that ordinary people without any technical background can compose their own services from a set of available services. To achieve this goal the service composition tool needs to be user-friendly and have the possibility to be used for many purposes.

The following concepts and models is used in the end-user composition tool:

Ubiquitous Computing: A model for human-computer interaction where information processing has been integrated into everyday objects. The devices will communicate with other devices and share information about its context. The devices run in the background without the need of human attention [1].

Service Oriented Computing (SOC): An approach to programming that is used when developing application that utilizes services as fundamental elements [2].

Service Composition: The concept of creating new services from already existing services by discovering and integrating them into a single service [3].

User-Centered design process: User-centered design is an iterative design philosophy and process, where the needs, requirements and limitations of the end-users of a product are given extensive attention through each stage [4].

Control flow-based compositions A Control flow-based composition approach represents a sequential composition process, where a task or service is required to be completion before the next task. This means that the control flow defines the order in which atomic services are executed [5]. Figure 1-1 shows how the control flow-based composition works:

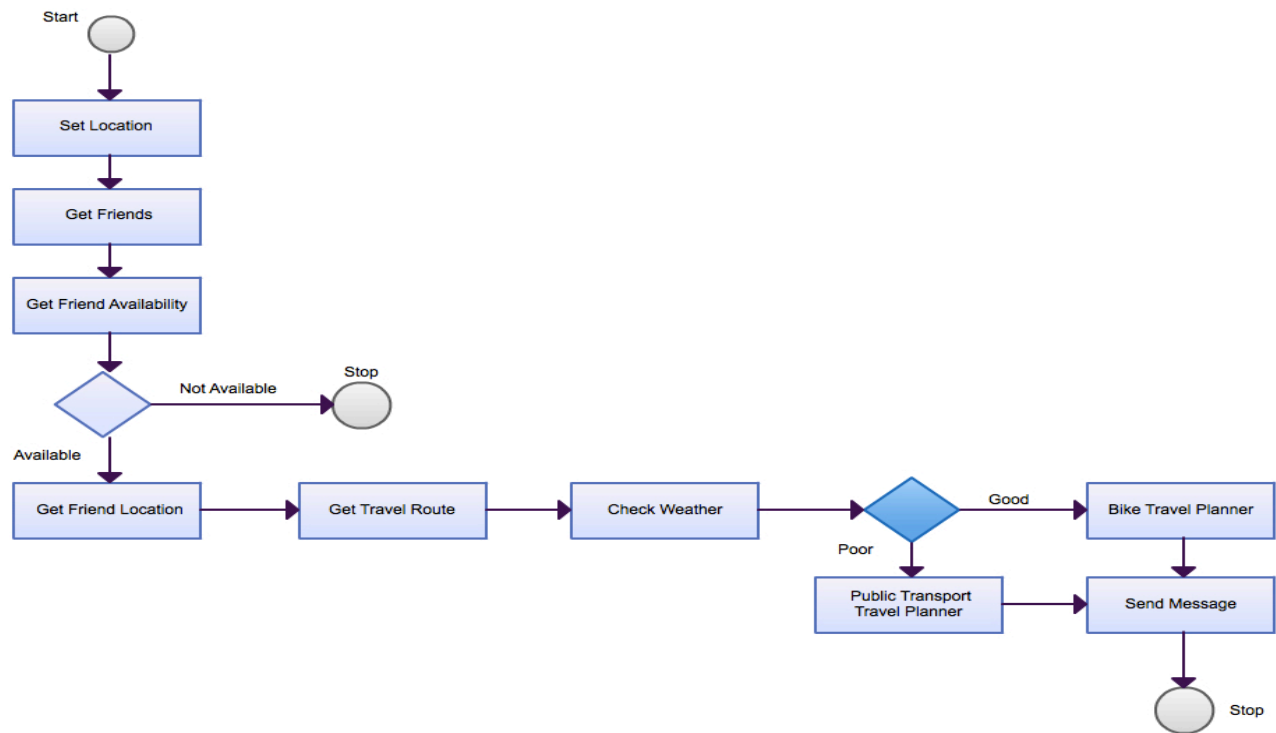


Figure 1-1: Control Flow-Based Composition (replica of the figure in [5])

1.2 Problem Statement and Objectives

The existing tool allows end-users to compose their own services from existing service components. It is usually the service-providers that compose these predefined services, but these services do not always meet the users requirements. The vision is that ordinary people without a technical background can compose their own services from a set of existing services while they are on the move. As previously mentioned the existing tool consists of two applications Easy Composer (JavaFX) and the EasyDroid (Android application).

The Easy Composer is an application written in Java FX, and started out as a service composition tool that enabled end users to make their own services using simple service components. With further development of the Easy Composer, calendar and map functionality was added. The latest version contains a Composition area, a Map and a Calendar. The idea is to create compositions from simple components and save it as a profile. You can then make use of this profile by setting it to a location on the Map or/and at a specific time in the Calendar. The profiles with location and time is stored in an XML format and then uploaded to an online server. The uploaded XML can then be downloaded to the users phone running the EasyDroid application. The EasyDroid will then run in the background and activate the profiles on a specified location or time. The Easy Composer and EasyDroid application will be described in more detail later in the thesis.

In this project period, the service composition tool has been further developed by me and another participant. Based on feedback from test users, we came to the conclusion that having two different applications with different functionality and interfaces was cumbersome. Therefore it was

decided to create a common interface for the end user, which combines the two applications functionality and could be accessed across platforms. Hence, the main task of this thesis was to develop and implement a new system, which combined the two existing applications functionality and also add new functionality to the tool. The Activities during the Master Thesis period is presented in the figure below:

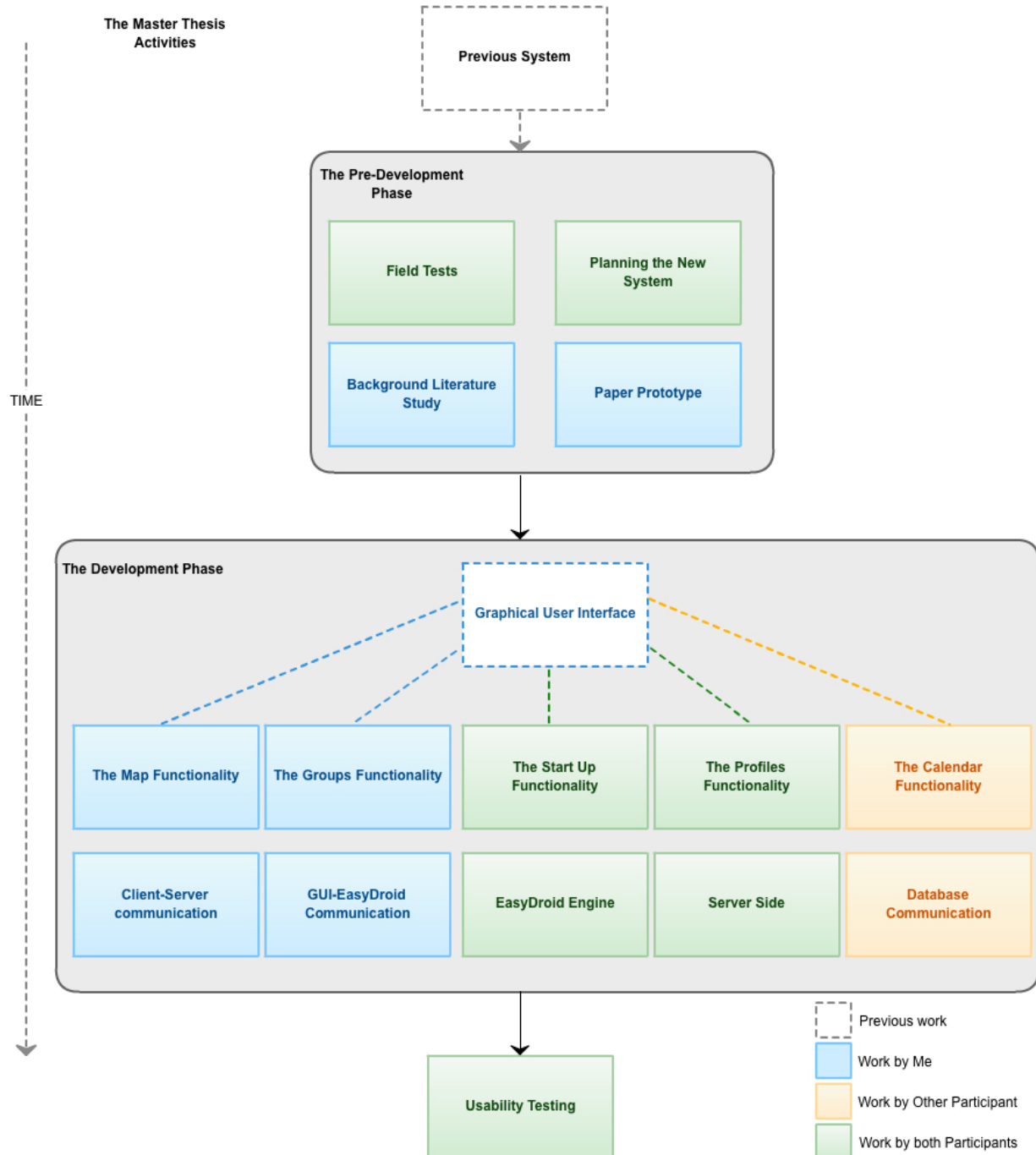


Figure 1-2: The Development process during the Master Thesis period

In Figure 1-2 the different activities carried out in the Master Thesis period are presented. The boxes have different colours in order to understand the work carried out by the other project participant, both and myself. As seen in the figure the blue boxes is work carried out by me, the

green is functionality both have been working on and the red is the other participants work. The top box in Figure 1-2 is the previous system developed. The second rectangle shows the activities before the development phase. In this phase meetings were held on planning the new system. The other activities were field tests and a background study on different literature regarding the new system. This resulted in a paper prototype being created.

The third rectangle shows the activities carried out in development phase of the new system. The new system contains a client side, a server side and the communication between. The client side contains a web-based graphical user interface (GUI) and a remake of the EasyDroid application. The server side contains the server code and a database. All the activities carried out on the GUI are showed with a striped line from the *Graphical User Interface* box. After the development phase was over, a usability test was carried out.

The project task included the following objectives:

1. **Develop the new system:** Propose improvements to the interaction mechanisms between end-users and the new user interface by using paper prototyping. This will result in a new graphical user interface(GUI) being developed and implemented. The functionality of the existing tools applications will then be implemented. The GUI will communicate with a remade version of the EasyDroid application, named the EasyDroid Engine, and a new developed server side including a database for storage.
2. **Add new functionality:** Develop and implement new functionality to the tool for the new system.
 - a. Integrate and implement new functionality and improvements proposed in my previous project to the new system:
 - i. The further development of the design and implementation for a location-based service, The Buddy List, and integrate it with the new application.
 - ii. Develop and implement a new component, that lets users access content on the Internet.
 - iii. Integrate and implement a mechanism for detecting service composition conflicts that resolves it in a simple manner.
 - b. Propose and develop new functionality to the tool.

These objectives will improve the usability and utility of the service composition tool. With utility, we mean what the tool can be utilized to. With usability, we mean the user-friendliness and usefulness.

3. **System Testing:** Carry out a usability-test with a focus group and field tests. Propose new scenarios that can be performed with the new system.

To carry out and complete these tasks, the existing service composition tool was studied. In addition, background literature was studied in order to develop the new system with a sufficient qual-

ity. Research was also done on finding the appropriate technology for the new application. Before and after the new system was developed, usability testing was carried out. The improvements to the system are based on end-user feedback from an on-going end-user evaluation process. This process is being performed within the scope of UbiCompForAll (a Norwegian research project in cooperation with Sintef) [6].

1.3 Structure of the report

Chapter 1 gives an introduction of the Master Thesis with the background of the previous system, motivation and the problem to be solved. Chapter 2 presents the theoretical background literature and related work. Chapter 3 gives a description of the previous version of the system. Chapter 4 is the main part where the project work with methodology, design and implementation is described. Section 4.1 presents an overview of the system being developed. Section 4.2 and 4.3 gives a description of the client and server side of the new system. Section 4.4 gives a description of the communication and interaction between the different parts of the system. Chapter 5 presents the system testing and results. In chapter 6, a discussion of the new system is presented as well as future work. Finally, Chapter 7 concludes the Master Thesis.

2. Theoretical Background and Related Work

In the Master Thesis a variety of topics that have been studied. Some of the topics are theoretical, while others are practical. Some of the practical topics for this Master Thesis are e.g. developing applications using Google Web Toolkit and developing for the Android platform. However, this chapter will describe the research on the theoretical background and literature of different fields related to the Master Thesis tasks. This background research is important to ensure the tasks will have a sufficient quality. In this chapter, the theoretical topics related to the thesis tasks will be introduced, in order to get a better understanding of the resulting design and implementation. First off, the topic of user-centered design is presented in order to understand how the previous system is developed and also the further development. Secondly, a brief descriptions of location-based services are presented, which relates to the development of the location-aware Buddy List System. Thirdly, the topic of conflict resolution in SDL and UML is presented, related to the proposed and implemented Conflict Resolution. Finally, the Google Web ToolKit (GWT) is described.

2.1 User-Friendly design

In order to make the end-user service composition tool popular and succeed in the public market, it is essential that the tool is user-friendly and meets the different user requirements. This was an important non-functional requirement for the new graphical user interface (GUI) being developed. To achieve this goal, the focus must be on the end-user.

2.1.1 User-Centered Design

The diversity of the “mass-consumer market” has a great potential for a widespread use of end-user composition. However, this creates challenges for design and usability. People have different needs, motives and technical background. The challenge is to find the right balance between simplicity and the functional richness of the composition environment. Developers with technical background may have a limited knowledge of how useful the composing of services is for end-users without a technical expertise. End-users also have a huge variety of needs. The best way for the developers to obtain enough information about the end-users needs and requirements is to actually study the end-users in a real usage environment, over a longer period of time [7]. One way of conducting this study is utilizing a user-centered design process. User-centered design is an iterative design philosophy and process, where the needs, requirements and limitations of the end-users of a product are given extensive attention through each stage. The different stages are illustrated in Figure 2-1:

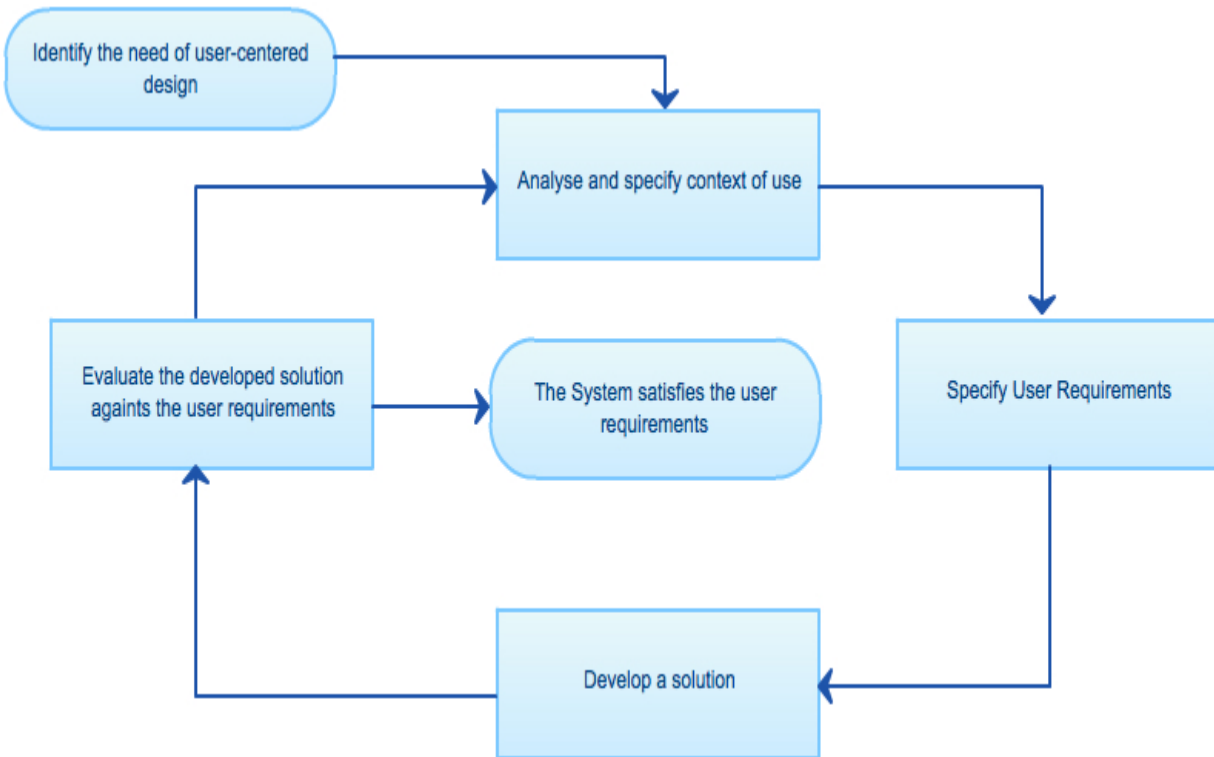


Figure 2-1: User-Centered Design Process

The two first stages are analysing the context of use and specifying the user requirements. The next stage is developing a solution for the requirements. The last stage is evaluating this solution by testing and see if it matches the user requirements. If the evaluation is successful the design process is finished, if not it repeats the cycle. The user-centered design process requires the designers to foresee and analyse how a fictional user likely would use the product, and then validate the assumptions by testing the behaviour of real users. The user-centered design has three main tools that is used for the analysing the context of use [4]:

- *Persona*: A fictional user with the characters of a real user is created from the user needs. The persona is usually created after a field research of the behaviour of group containing primary stakeholders [8].
- *Scenario*: A sequence of events where the *persona* is the main actor. The problems and needs of the *persona* are visualized in the events. It is particularly useful for removing focus on technology to open up design possibilities [9].
- *Use-Case*: Related to the scenario, but a more low-level approach. The interactions and details are more intricate [4].

User-centered designing has been used in the preceding project phases where the development of scenarios has had the main focus. Scenario building is a widely accepted way to generate design ideas for new systems and products and to identify the possible users and contexts of use for these systems and products [4]. Furthermore, this project also utilizes the user-centered design process. In the initial phase of the project the user-centered design process has contributed to under-

stand the user needs and requirements. The user-centered design process will contribute to having a sufficient level of utility introducing new functionality to the system, and also improving the usability of the system.

2.1.2 Usability Requirements & Measures

Designing a GUI with quality features such as usability and usefulness, careful planning is required. Also, sensitivity to the user needs, proper requirement analysis and sufficient testing is important factors for succeeding. In a successful designed system the interface almost disappears, enabling the user to focus and carry out the tasks effortless. To achieve this the designer needs to set up goals and requirements to the system [10].

The first goal of doing requirements analysis is to determine what tasks the user will be carrying out. Normal and frequent tasks is usually easier to determine than exceptional tasks and repair tasks dealing with system errors. To determine these tasks is very important, since the user will easily get frustrated if the functionality of the system is inadequate. Another common mistake is making the GUI too complex with excessive functionality. This will affect the simplicity of usage, complexity of implementation and learning of the system in a negative manner [10]. Another important requirement is the reliability of the system. The actions should work as specified, data and updates should be displayed correctly. Misleading data or unexpected results may make user lose its trust in the system. Also, the software, hardware and network must have high availability. An unavailable system will most likely create errors and making the end-user lose its willingness to utilize the system [10].

The third set of requirements is to consider which context the system is used and promoting appropriate standardisation, integration, consistency, and portability. The designer needs to decide if the changes or improvements offered are useful enough to compensate for the time and retraining with the end user. Even small changes in software interface increase the learning time [10]. For this project the applications consistency and portability are the most important features. Consistency within an application refers to common terms, objects, action sequences, buttons, colours etc. Portability refers to potentially share data and GUI across platforms. For the developing the new GUI this is an important requirement, as the end user will interact with the GUI on different platforms. This could pose a challenge for the designer, since different platforms will have different display sizes and resolutions, pointing devices, data formats etc. The display sizes and pointing devices especially need to be emphasized when developing the GUI.

After the usability requirements have been determined, the developer can focus their attention on design and testing purposes. Determining the user base and the set of tasks the users will do is the basis for establishing usability requirements and measures. Establishing measurable objectives for each task is important for guiding the developer. The goals of efficiency and user satisfaction can be evaluated with the following measurable objectives [10]:

1. *Time to learn*: The time it takes for a typical user to learn how to use the system to carry out the relevant tasks.
2. *Speed of performance*: The time it takes the user to carry out the relevant tasks.

3. *Rate of errors by user:* The number and which types of errors the users generate when carrying out the tasks.
4. *Retention over time:* The maintaining of knowledge of the tool over a period of time. How long the user remembers how to carry out the tasks. This may be closely linked to time to learn and how frequent the system is used.
5. *Subjective satisfaction:* The subjective opinion of using various aspects of the interface.

It is difficult to succeed in all of the objectives above, since there are often trade-offs between the objectives. Having a very low error rate in a system may affect the speed of performance, making it slower. Different applications will have different keys determining the success. For some applications it could be the time to learn and others it could be the subjective satisfaction [10]. It is important that the developer is aware of the trade-offs and also is aware of the different needs and requirements of different user demographics. The actual design of the GUI can start when the developer has determined this. Before the developer starts to design it may be a good idea to do some research on common design guidelines and principles that has already been proven to be successful.

2.1.3 Design Guidelines & Principles

Design guidelines help the developer to create a system that is consistent in terminology, appearance and action sequences. Numerous of guidelines has been written, and critics often complain that the guidelines are too specific, hard to apply and incomplete [10]. Hence, this section will address some key topics regarding design guidelines rather than choosing a specific one. Only the guidelines those are relevant for this project is chosen.

The first topic we consider is navigating the interface. A sample of widespread applied guidelines is described here [10]:

- *Standardize task sequences:* The user should perform actions in the same sequence and manners in similar scenarios.
- *Use unique and descriptive headings:* The headings should be unique and describe its content.
- *Use check boxes for binary choices:* Provide a check box whenever the user has a choice with two states.

The sample of guidelines above comes from the National Cancer Institute effort to aid government agencies with designing informative web pages. These guidelines have a widespread usage and are supported by a variety of research studies [10].

The second topic regards the organization of the display. This means how we organize the layout of items displayed to the user. Here, a sample of different objectives relevant to the Master Thesis is presented [10]:

- *Consistency*: The terminology, abbreviations, formats, colours and capitalization should be standardized.
- *Minimal memory load*: Information should be presented so the user is not required to memorize information. The completion of a task should be done in a minimal number of steps.
- *Flexibility of user control*: The user should be able to get the information it needs in the most convenient way. For example, this could be sorting the rows in a table based on a column.
- *Presentation of data*: Present data only if it assists the user.
- *Involve the user*: Involve test users in the design process to get feedback.

These objectives are taken from the Smith and Mosier (1986) five high-level goals for data-display guidelines and Lockheed (1981) generic guidelines [10]. The objectives are a good starting point for the designer but it needs to be expanded specifically for each application.

Before the design phase begins the developers should have an understanding of the intended users. This is a never-ending process as there is a great deal of information to be gathered and the users keep changing. To differentiate between users, one could categorize the users based on the user technical skills and experience with the system. An example of a categorization is dividing into novice first time users, knowledgeable intermittent users and expert frequent users. For the first time users, functionality with instructions, dialog boxes and informative feedback would be helpful. For the intermittent users an interface with structured menus, consistent terminology and action sequences would make it more satisfactory. Finally, the experts would probably be best off with a systems with rapid response times, brief feedback, and shortcuts to carry out actions [10]. Here the focus is to carry out the tasks quickly, and an unnecessary feedback and instructions would only frustrate the user.

The design of the GUI becomes increasingly more difficult when more than one the categories represents the systems target group. The strategy for designing a system with multiple category usage is called a multi-layer system [10]. A multi-layer system allows for differentiating between users with varying levels of knowledge. This could for example result in that a first time user would be presented a slightly different interface than an expert user.

After determining which user category the system falls under, the developer needs to identify the tasks of the user should carry out. The task analysis is usually done by interviewing and observing the users. This will give the developer a good understanding of the frequency and sequence of the tasks and also helps in choosing what tasks should be supported. The developer should choose these tasks carefully, since including all possible tasks could clutter the system. Tasks

that are carried out frequently should be done simply and quick [10]. Basically the analysis of how frequent the tasks occur should decide the accessibility of the task.

Finally, this section will present eight principles of interface design called the “golden rules”. These principles are applicable for most interactive systems and are based on experience from previous studies of interface design. The principles have been redefined for over two decades [10]:

1. *Strive for consistency*: The objects in the system should be consistent. This includes that action sequences in similar conditions are consistent, identical terminology is used in menus and prompts and consistent use of fonts, colours, icons, layout, names etc.
2. *Cater to universal usability*: The system should recognize the needs of the different users and accommodate for them by adjusting the features presented relative to the users knowledge. For example, explanatory help features for first time users, and shortcuts for the expert users. This would most likely improve the perceived system quality for all users.
3. *Offer informative feedback*: There should be some sort of system feedback for each user action. Frequently and minor actions should have brief feedbacks and infrequent and major actions should have a more substantial feedback.
4. *Design dialogs to yield closure*: Make sequences out tasks and organize them to show progression to the user. The completion of a sequence of actions should give informative feedback.
5. *Prevent errors*: The system should be designed in a way that the user cannot make serious errors. For example, making menu items that are not appropriate inaccessible. The actions that cause errors should leave the system in the same state or give instructions about restoring the state.
6. *Permit easy reversal of actions*: The actions done to the system should be reversible. This relieves the users anxiety, since the user knows that the erroneous actions can be undone.
7. *Support internal locus of control*: The user should feel that he is in control of the interface. Surprising interface actions and inability to execute the desired action causes dissatisfaction and anxiety
8. *Reduce short-term memory load*: Minimize the short-term memory load required by the user. The human mind has limitations processing short-term memory information. Hence, the display should be kept simple and navigating to new displays should not require information to be remembered from other displays.

These rules need validation and special tuning for specific designs. In other words, the principles must be refined and extended for the particular environment [10]. This section has presented guidelines and principles for designing the GUI. In the next section, a method called “Paper Prototyping” for starting the actual design of the GUI is presented.

2.1.4 Paper Prototyping

Paper prototyping is a great starting point for developing GUI's for web sites, web applications and other software that requires human-computer interaction. Before you actually implement the user-interface the developer draws a sketch of the GUI on pieces of papers, based on the tasks the developer wants a user to accomplish. This includes windows, menus, pop ups, text fields, buttons etc. After this is done a usability test with other developers playing the computer are conducted. Under the usability tests the developers are given tasks to manipulate the pieces of paper simulating how the GUI would behave. Then test users are given realistic tasks, based on scenarios where they interact directly with paper prototype. A team of observers takes note on how the test-users perform and react. Since no aid or help are given, the developers get a good understanding of what parts of the interface that are confusing and self-explanatory [11].

Paper prototyping is good for detecting a variety of GUI design problems. Problems related to terminology are a common user-interface issue. The terms that are used should be understandable and self-explanatory. Another common problem is the navigation or workflow of the interface. The sequence of steps should match what a user expects, and the number of steps should be kept to a minimum. The page layout and content should also be presented such that the user gets the sufficient information of how to make decisions, but not frustrate the user. Finally, missing functionality is easily detectable when conduction paper prototyping. Paper prototyping is especially useful for detecting the problems stated above. However, paper prototyping isn't always the ideal. Paper prototypes do not demonstrate technical feasibility. It is possible to create paper prototypes that can't be implemented. Paper prototyping also doesn't show the response time of the system [11].

One of the main benefits by doing paper prototyping is that you can check if you are on the right track before you implement anything. During a usability test you can learn new issues that can have a huge impact on the GUI. With paper prototyping you can make fast changes, which would have been cumbersome if the interface was already implemented. Even more substantial changes only will only take a couple of hours. There are also some concerns related to paper prototyping. One of the concerns is if the paper prototyping are missing important problems or finding false ones. However, according to [11], scientific studies and experience indicates that this is not an issue. Paper prototyping is a good way of finding significant problems early in the design process, and also lets you find a good solution before you invest time in implementing it [11].

2.2 Location based services

Location based services are services that compute the location of an entity and utilize it in an application or service. Location based services have actually existed for over 20 years, and was first introduced with the indoor location system “Active Badge” utilizing infrared technologies. For outdoor location sensing, the Global Positioning System (GPS) has become the most popular. The introduction of Smart phones with location-sensing technologies has made a huge increase in the number of location-based services being developed [12].

2.2.1 Location-sensing

Sensing an entity’s physical location can be done in many ways and different solutions have been developed. They differ with respect to accuracy, coverage and frequency and also the cost of installing and maintenance [12]. We can divide the location sensing systems into coarse-grained and fine-grained systems, where coarse-grained systems give a rougher estimate of the location than a fine-grained system.

The EasyDroid application is currently only available on the Android platform. The Android platform supports two kinds of location-sensing mechanisms, a coarse-grained network-based mechanism and the more fine-grained GPS mechanism. The network-based mechanism uses triangulation by collecting information about the cell-towers or Wi-Fi hotspots and eventually calculates the location from this data [13]. The GPS mechanism calculates the position by measuring the difference in the arrival of satellite signals [12]. The mechanisms have different accuracy and energy consumption [13]. Hence, the network-based mechanism uses less power than the GPS mechanism, but are often less accurate in an outdoors environment. However the network-based mechanism will be more accurate indoors and in high-rise urban areas [12]. Active use of location-sensing applications will drain the battery of the device [13]. When designing our location-based service we need to define the requirements for accuracy and energy consumption in order to choose the optimal mechanism.

2.2.2 Google Latitude

Google Latitude is a location-based service for Smart Phones developed by Google. The user’s location is plotted in Google Maps and it allows for certain users to see the phone user’s location via their Google latitude account [14][15]. The location sensing works as a combination of global positioning satellites, Wi-Fi and cellular tower triangulation. There are many privacy concerns with Google Latitude because of the sensitive nature of the geo-location data. Therefore Google Latitude lets you customize what location content that is available to the other users [15].

In the previous project a location aware application named the Buddy List was developed. In the new system the functionality of the Buddy List is integrated. This functionality has its similarities to the Google Latitude application. The combination of GPS and network triangulation is an important feature that is applicable to the Buddy List functionality. Furthermore, the privacy concerns with Google Latitude also applies for the Buddy List.

2.3 Conflict resolution

The control flow-based composition approach has many similarities with the Specification and Description Language (SDL). SDL is an object oriented, formal modelling language intended for the specification and description of telecommunication systems and protocols [16]. Hence, for the conflict resolution in the composition area, the concepts of SDL validation are useful for developing a conflict resolution. Here are some conflicts or errors that are commonly examined by validation and testing that applies to the service composition tool [16]:

- *Deadlock conditions*: The composition enters a state that cannot be left due to a missing or occupied resource.
- *Livelock conditions*: The composition enters cycles that cannot be left due to a missing or occupied resource.
- *Liveliness*: Each component of the composition can be reached from the initial state.
- *Termination*: The final state – or an idle state for cyclic systems – can always be reached.
- *Recovery from Failures*: The system can recover to a normal state within a limited time after an error has occurred.

In the late 1980's, AT & T developed a tool for validating SDL called SDLvalid. The structure of and usage of this tool is illustrated in Figure 2-2:

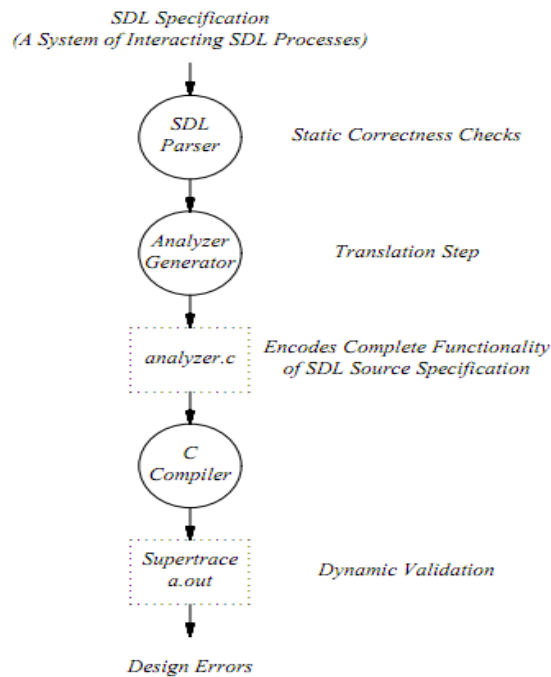


Figure 2-2: SDLvalid Structure Overview

This tool checks for design errors made by the end user. The design errors relevant for the new system is the following [17]:

- Incompleteness: That a message sent from one component has a receiver.
- System Deadlock: Previously explained for the Deadlock condition above.
- Unreachable components. Components have no connectors connected to it.
- Violation of (user specified) Correctness assertions: Components that is not supposed to be connected is connected.
- Violation of (user specified) Temporal Logic formula: The composition does not have the correct behaviour compared to the Temporal Logic Formula.

The first item is checked statically, during the parsing phase. The rest of the items require dynamic checking by using reachability analyses [17]. The last two validation checks are rarely used, but they are highly relevant for the service composition tool.

2.4 Google Web Toolkit

The Google Web Toolkit is an open source toolkit for developing sophisticated browser-based AJAX applications. GWT simplifies the process of making the web applications with JavaScript, XMLHttpRequests etc. In GWT, you can develop the application entirely in Java programming language. The Java is compiled into highly optimized JavaScript that can run across all browsers, including browsers for Android, Iphone and WinPhone. Hence, anything you can create with JavaScript and a browser's Document Object Model (DOM), can be created with GWT. Also, interacting with external written JavaScript is possible [18]. Developing AJAX applications becomes much more effective since you have a higher level of abstraction. GWT also allows for fast debugging as you don't need to compile the code each time you have made a change. The changes are viewable instantly in the browser as it would be with JavaScript, but with the possibility to inspect variables, set breakpoints etc. The GWT compiler optimizes the code during compilation by in-lining methods, removing dead code, optimizing strings etc. [18]. GWT also provides the possibility to visually style the GWT applications using cascading style sheets (CSS).

2.4.1 Communicating with server

When the application needs to interact with a server, it will make a HTTP request across the network. GWT supports a couple of different ways for communicating with backend servers via HTTP. One option is to use the GWT Remote Procedure Call (RPC) framework to make calls to Java Servlets, or another option is to use the GWT's HTTP client classes to send and receive custom HTTP requests. The GWT RPC framework is the most efficient way, and it enables the user to send serializable objects across the network [19].

The biggest difference between using AJAX and traditional HTML web applications is that AJAX applications do not need to fetch new HTML pages during execution. AJAX applications runs like an application inside the browser and don't need to request new HTML pages when updating the GUI. When the AJAX application is receiving or sending information to a server it uses a mechanism called making a remote procedure call (RPC). The GWT RPC makes it easier to send and receive objects over HTTP. By moving all the GUI logic to the client, it will reduce bandwidth and load on the web server. This will result in greatly improved performance and a more fluid user experience [19].

Making a remote procedure call is often referred to as invoking a service. The diagram under shows a nice illustration of the moving parts the GWT RPC is set up.

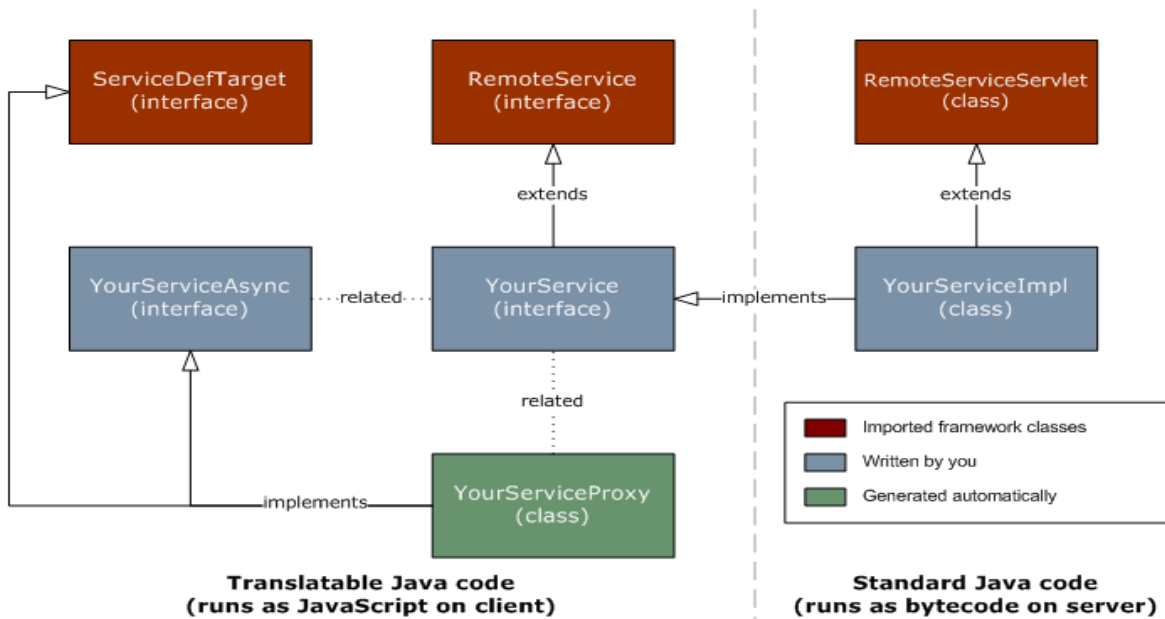


Figure 2-3: GWT RCP

From Figure 2-3 we see the Service Proxy as a green box. It is automatically generated, and is invisible to the user. The blue box in the middle is the Service component interface. It is the interface where the users define which methods that can be called and implemented in the server side. The blue box to the left is the Service Asynchronous interface. It takes care of the asynchronous callback response from the server and is implemented on the client side. When a client invokes a service on the server, it makes an asynchronous method call. This means that the Service methods need to pass in a callback object that is notified when an asynchronous call completes. This way the caller is not blocked until the call completes [19].

On the server side, the Service Implementation is where the implementation of the methods defined in the Service Interface resides. When a client request is issued, the server implementation does some processing in order to send a response. This is based on well-known servlet architecture [19].

2.4.2 GWT JSNI

For gaining access to functionality not exposed by the GWT class API's you need to integrate the GWT with external hand written JavaScript. GWT provides a feature called JavaScript Native Interface (JSNI) for doing this by allowing you to integrate JavaScript directly into the applications Java source code [20]. JSNI can be used in many ways: Implement Java methods directly in JavaScript, making calls from JavaScript code into Java and opposite, read and write Java fields from JavaScript etc. [20]. For accessing native Android code functionality the JSNI feature is especially useful. However this should be used carefully as writing bulletproof JavaScript is sometimes notoriously tricky. Furthermore, JSNI can be less portable across browsers with memory leaks and making it harder for the compiler to optimize the code [20].

3. Previous System

The main focus in the development of the previous system was on the existing service composition tool applications Easy Composer and EasyDroid. The service composition tool made it possible for end users to compose their own services and use them in varying scenarios. The tool is a part of the UbiCompForAll project, which is a research project in cooperation with Sintef. The tool has been in development for 2-3 years. This chapter describes the two applications Easy Composer and EasyDroid, how they communicate, and the development until the start-up of the Master Thesis period. The chapter begins with describing the Easy Composer application in section 3.1, followed by a description of the EasyDroid application in section 3.2. Section 3.3 describes the development of the Buddy List system.

3.1 The Easy Composer

The Easy Composer is an application written in Java FX, and started out as a service composition tool that enabled end users to make their own services using simple service components. With further development of the Easy Composer, calendar and map functionality was added. The latest version contains the features Composition area, Map and a Calendar. This section presents the latest version of the Easy Composer.

By creating the services on the application level, it removes the reliance to the service provider/operator. In other words you can make use of the services independent of service provider/operator. The existing components have been developed in terms of telephone services. So the composing of components is actually creating basic telephone services. A composition and a profile is essentially the same, but the definition “profile” is used for a saved composition. To get a better understanding of the Easy Composer, we have to look deeper into how compositions are being made.

3.1.1 The Composition area

In the earlier versions of the Easy Composer, there were over 15 different components to choose from in the Composition area. However, having that many components made the application too complex and far less user-friendly. To improve the usability, the number of components got cut down to 6 basic components. Figure 3-1 shows the latest version of the Composition Area in Easy Composer. When creating a new service, the starting point is the Composition Area.

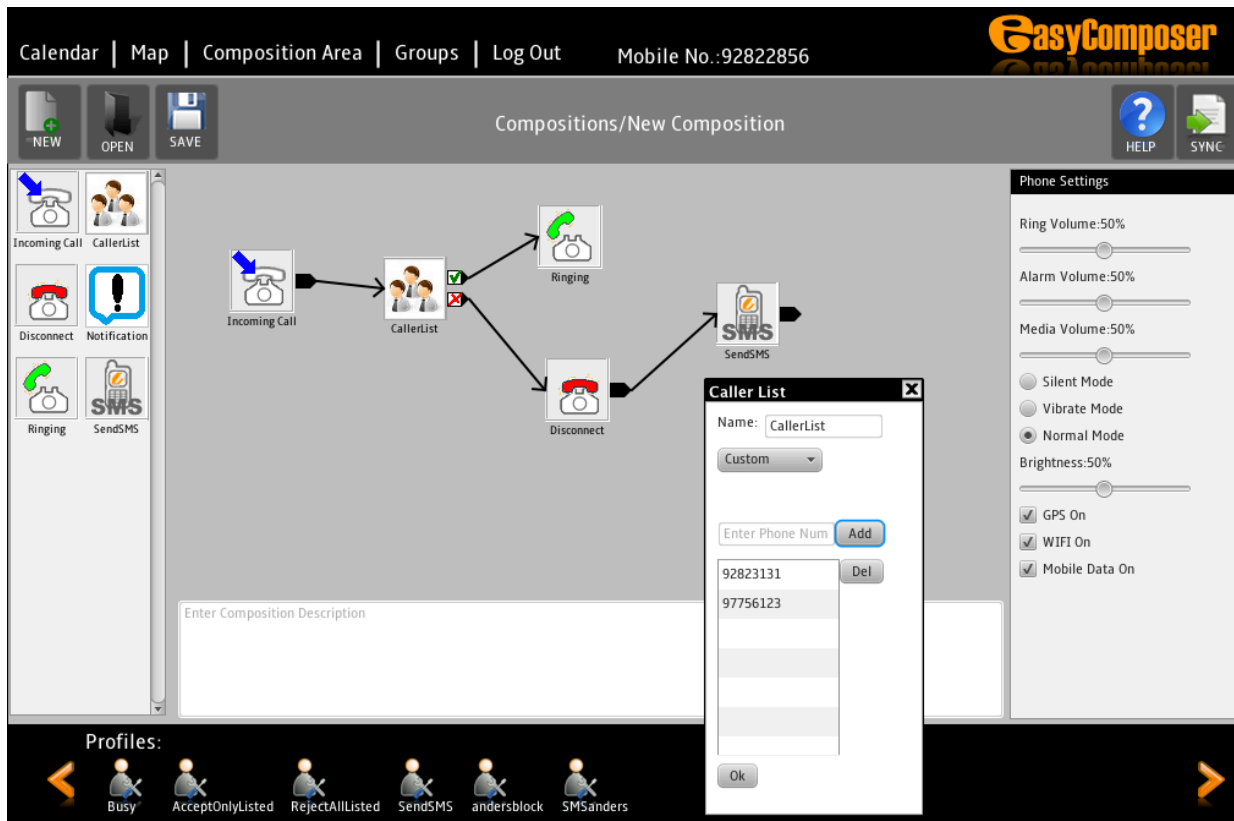


Figure 3-1: Composition Area Easy Composer

On the top menu you can choose between three icons that lets you create a new composition, open an existing composition or save a composition. On the right side you can choose the appropriate phone settings for the composition you are creating. On the bottom menu, you can choose from already existing compositions. Finally, on the left side there is a menu with six different components. Each of these components resembles a telephone service. The six different components are:

- *Incoming Call*: Represents an incoming call.
- *Caller List*: This is a list of numbers that can be used as a filter for incoming calls. For instance, the *Caller List* can either make a blacklist by having the numbers in the caller list disconnected, or a V.I.P list where only the numbers is in the list are connected. The dropdown menu also lets you choose predefined groups of phone numbers.

- *Disconnect*: Disconnects the incoming call.
- *Notification*: Notifies the user of actions that has happened in the background. For example if a call has been disconnected the user will get a notification of the event that has happened.
- *Ringin*g: Connects the incoming call, making the phone starts ringing.
- *SendSMS*: Sends an SMS to specific numbers. For instance in the event of a phone call is disconnected, one can use the SendSMS component to send back a message to the disconnected user.

By adding and connecting these components, we can make a huge variety of different profiles. The connections are represented by arrows, which show the flow of service creation. The composing of components is based on a control flow-based composition approach described in section 2.3. This was the selected approach because it was easy to use, understandable and most applicable for composing telephony services [21].

To get a better understanding of how a profile can be made we take another look at the figure of the Composition area, where a profile named “AcceptOnlyListed” is created. First the Incoming Call component is connected to a Caller List. Here the incoming call goes through a filter. The Caller List has two outgoing connections where the green “check” symbol represents the phone numbers that are in the list and the red “cross” symbol represents those who are not. Then we connect the green connection to the Ringing component, and the red to the Disconnect component. If the incoming call is in the Caller List, the phone call is connected, if not, the phone call is disconnected. Sometimes we want the disconnected callers to get a message reasoning the disconnection, so we add a SendSMS component and connect it to the Disconnect component.

After the profile is created, we want to use it for doing a particular activity. To activate the profile, the profile needs to be added to a time or location. This can be achieved by saving the profile, and add it to the Calendar or Map functionality.

3.1.2 The Calendar

Introducing the Calendar and being able to add profiles to it, created a new dimension for the Easy Composer and profiles could be used for a variety of scenarios. By adding the calendar functionality and profiles, the end users had a setting where they could make use of their composed services. The end users could make use of the profiles and incorporate it in their daily events. This helped enriching the user experience and services. Since the Calendar was first introduced it has undergone some improvements in terms of usability. The latest version of the Calendar function is shown in Figure 3-2 below:

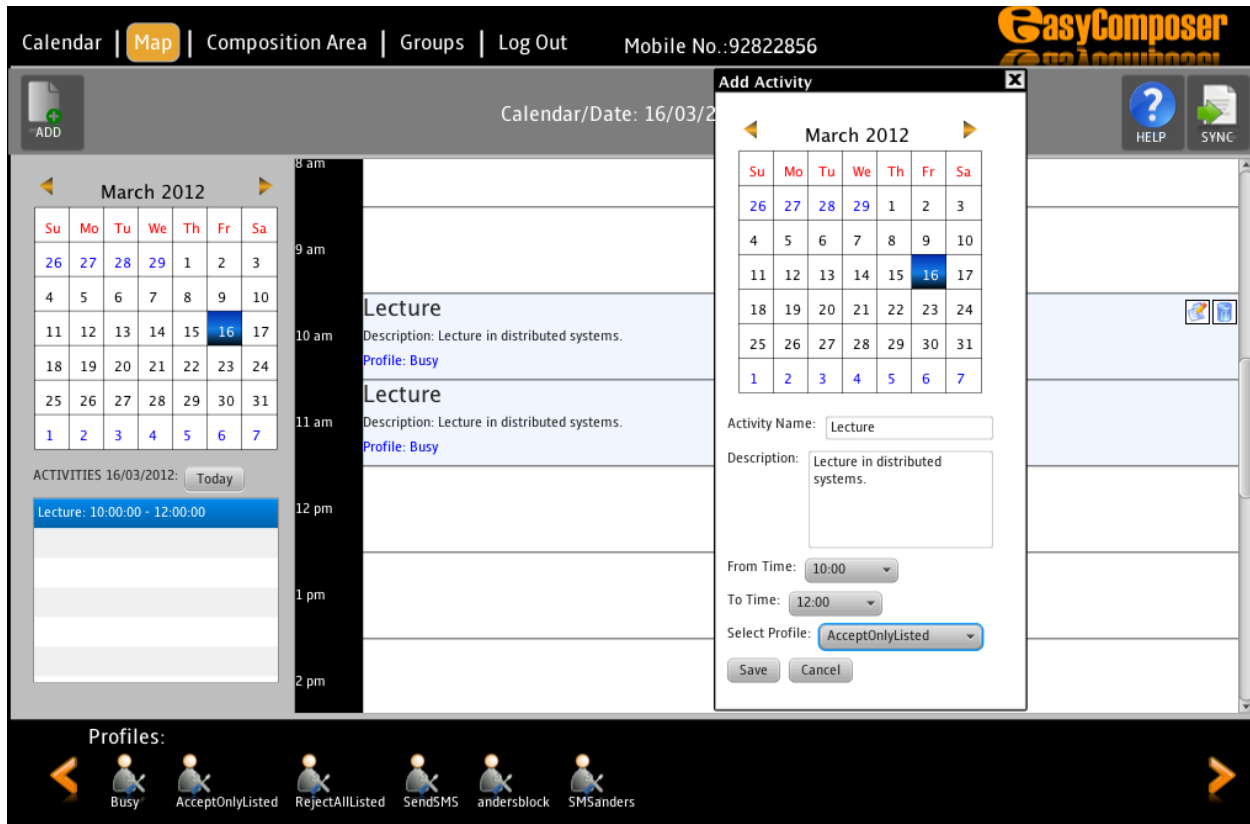


Figure 3-2: Calendar, Easy Composer

Here we can choose a specific date on the left, and then a window with the chosen day is shown in the main window. When the date is chosen, an activity can be added to the date. The activity is given a name, description, time period and profile. The most important feature is that you can add a profile to your activity. As shown in Figure 3-2, the profile we created in the Composition Area is chosen and also the time for when the profile is activated is set. Now the activity named "Lecture" has been created. The time is set from 10.00 to 12.00 on Friday, and the AcceptOnlyListed profile is added to it. With the Calendar functionality, the user could make use of the profiles in a context. To add even more context for the user, the profiles should react on the users location as well.

3.1.3 The Map

Adding the Map functionality, with users location context, increases the number of different scenarios that the profiles could be used in the Easy Composer. The Map functionality is based on Google Maps and it uses a search field, where you can search for addresses. When the search is done, a marker is added to the map. The Add function lets you create a new location with a name, the selected profile and proximity at the location on the map. The proximity is the radius from the point where the profile will be activated. Figure 3-3 shows the creation of a location named “Samfundet” and the AcceptOnlyListed profile being added to it:

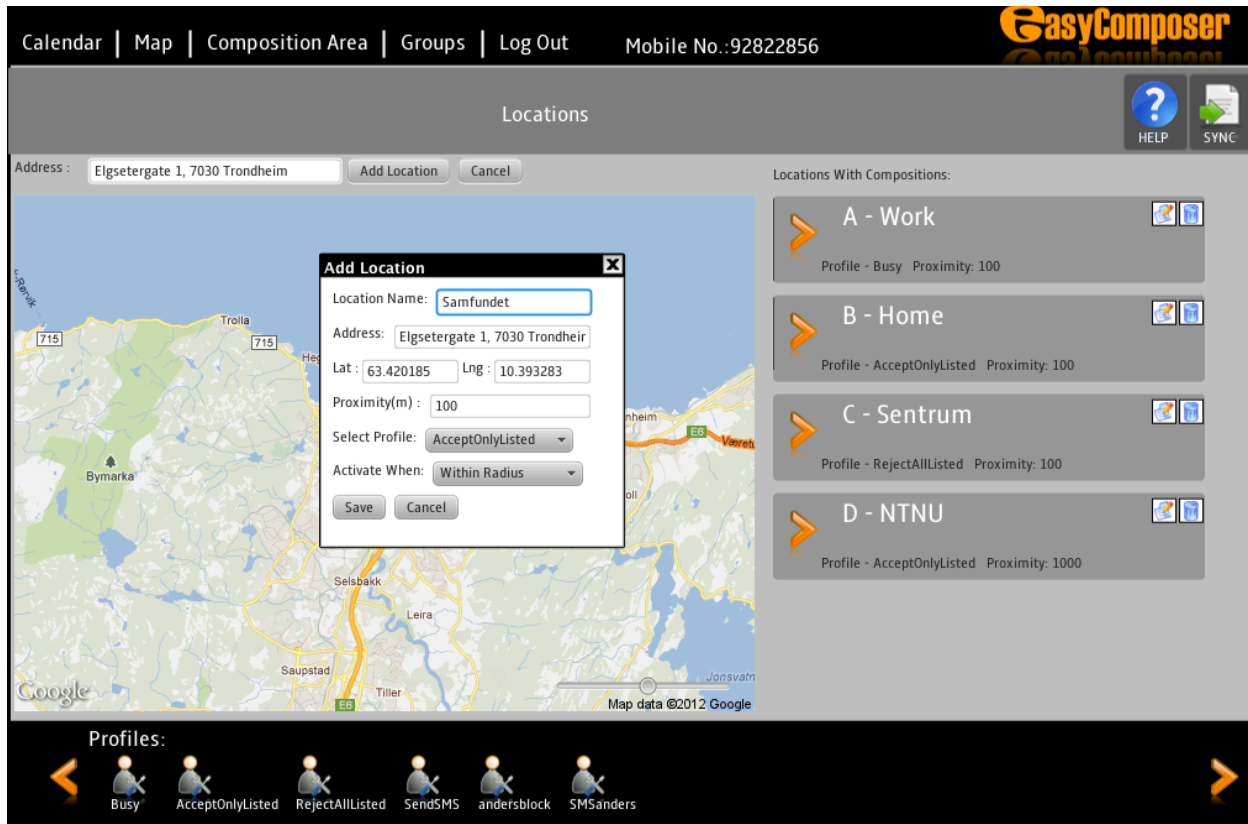


Figure 3-3: Map, Easy Composer

Now the profile AcceptOnlyListed profile is activated based on location and time. The user can use the export function to upload the XML to an online server, and then be downloaded to a users phone running the EasyDroid application. The XML is exported with the user information in order to download the correct XML.

3.1.4 The Groups

The latest addition to the Easy Composer is the Groups functionality, which lets you add contacts by phone number and put in one of the Groups three default groups: Co-Workers, Family and Friends. We can also create new groups if this is desirable. With the Caller List component in the Easy Composer we can choose a specific group and then retrieve the numbers in this group. Dividing the users into groups is necessary for giving the different users different functionality based on their relation to the end-user itself. The figure below presents the Groups GUI:

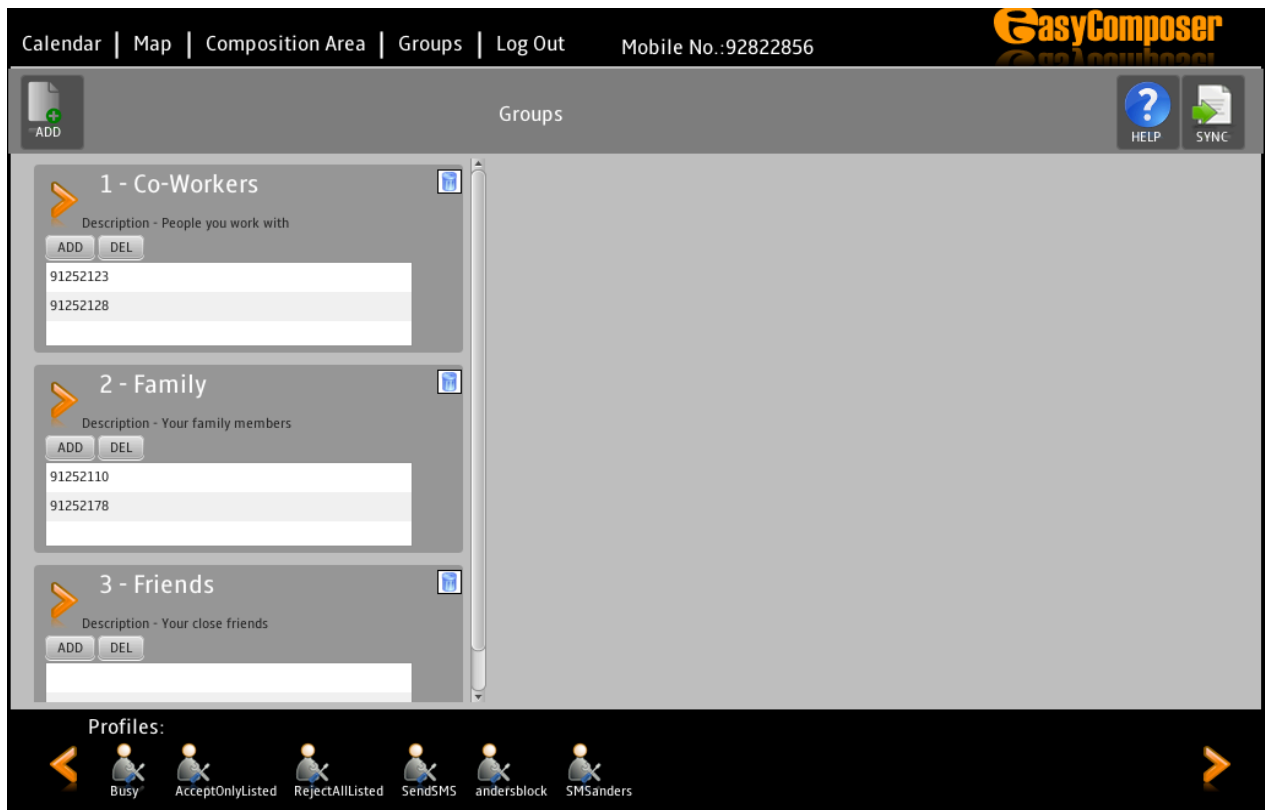


Figure 3-4: Map, Easy Composer

From Figure 3-4, we can see phone numbers added to the different groups. By pressing the Add button you can create new groups. The next section describes the EasyDroid application of the previous system.

3.2 The EasyDroid

The EasyDroid is a mobile application, which is developed for Android smart phones. EasyDroid utilizes the profiles by downloading the profiles from the same online server as Easy Composer upload profiles to. The profiles are downloaded in a XML format that consists of a composition of components with phone settings, and a time and location where the profiles are being activated. The EasyDroids architecture and how different parts work together is shown in Figure 3-5 [21]:

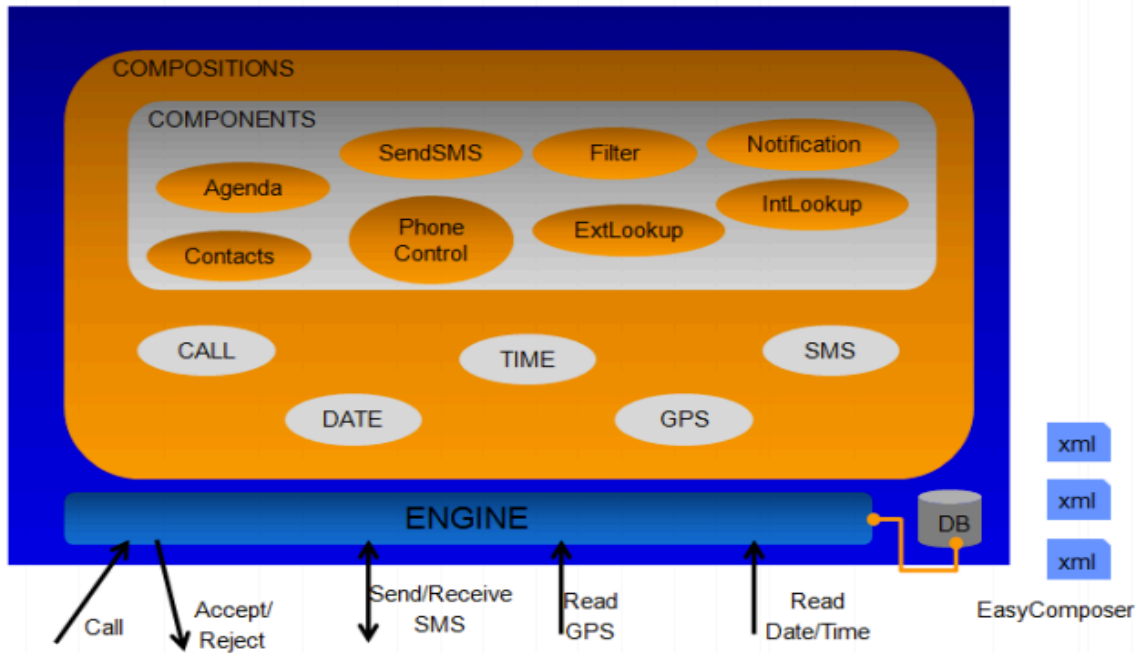


Figure 3-5: EasyDroid architecture

In addition to these components, a phone settings and location component has been added. The phone settings component is meant to capture the settings that a users want o be active. The location component gets inputs with locations and location proximities and is meant to enable certain services to be activated when the user is within the proximity.

The components of a profile make use of the services available in the Android environment or the Android software stack. For a location-based profile such as our AcceptOnlyListed profile, it would utilize the GPS, SMS, Time, Date and Call services. The Engine controls how the services are invoked and coordinated. When the XML with compositions activated by time and location is downloaded, the engine should be able to coordinate the different compositions to run at the time and locations specified in the XML [21]. The AcceptOnlyListed profile example showed in section 3.1.1 will then be activated either by sensing the location with the GPS service and check if the location is within the radius of activation set in the Easy Composer map, or using the DATE and TIME services to check if the profile is set for activation at a particular time. Figure 3-6 shows the GUI of the EasyDroid:

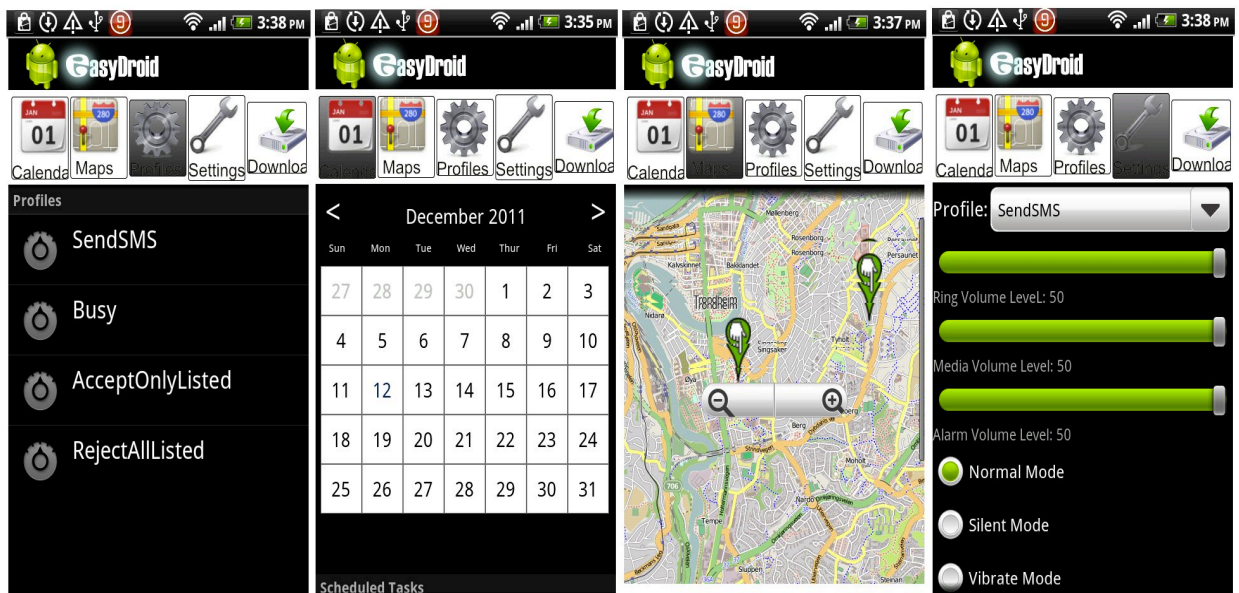


Figure 3-6: EasyDroid GUI

From Figure 3-6 we see the different parts of the EasyDroid GUI. In the most left and first screenshot the different profiles created in the Easy Composer is displayed. The second left screenshot shows the Calendar with dates the profiles are activated. The third left screenshot presents the Map showing markers where profiles are activated. Finally the fourth screenshot displays the phone settings related to which profile we choose. Here there is a possibility to manually configure the phone settings, if the phone settings are set wrong in the Easy Composer. The next section presents the Buddy List System developed in the previous project period.

3.3 The Buddy List

In this section a brief description of the Buddy List system that was developed in the previous project is presented. The Buddy List was never fully integrated with the previous system.

The Buddy List is a location-aware application developed for Android that sends user location updates to a server, and can also receive users location updates for the people in the list from the server. The Easy Composer make use of the updates sent to the server, with the feature Groups. The system is divided into the client side, which resides on the Android device, the server side where the Easy Composer resides, and the communication in between. A simple overview of the system is shown in figure 3-7:

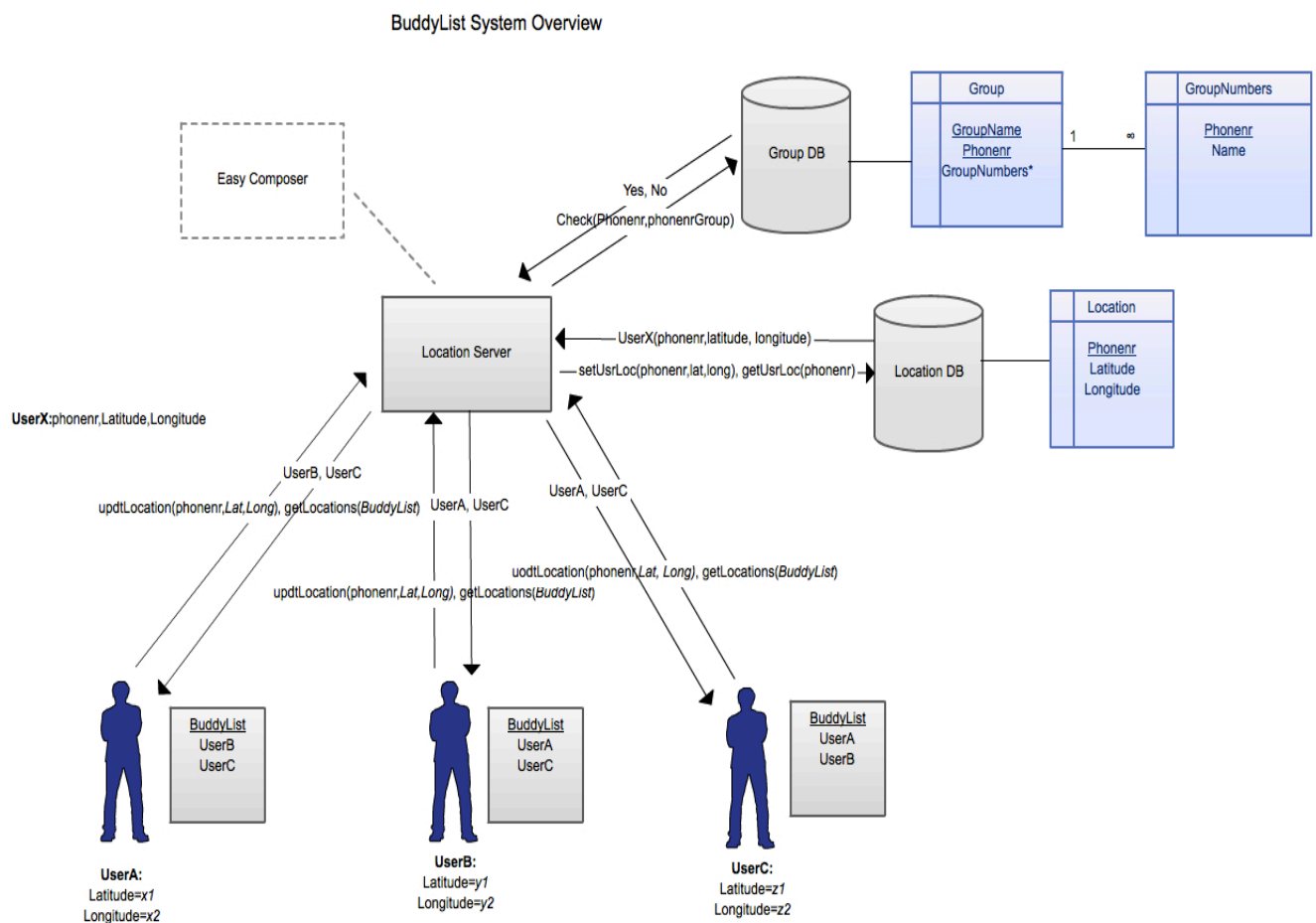


Figure 3-7: Buddy List System Overview

The different end-users clients calls the *updtLocation()* method, which sends a XML formatted message containing the users phone number and the location containing latitude and longitude. The location server receives the message and updates the user database using the *setUsrLoc()* method. The end users can also request the server for the locations of the users in the Buddy List by calling the *getLocations()* method. The *getLocations()* method sends a XML formatted mes-

sage containing the phone numbers of the users in the list. In order to get these listed user locations the users must have the requesting user in their group. This works like a simple authorization mechanism. Hence, the Location Server does a simple check for this, before it queries the Location database for the respective users locations. Finally the Location Server sends the listed users location updates back to the end-user client as a XML file. The Easy Composer is represented in the figure with striped lines.

The first thing a user will see when executing the Buddy List application is the user interface showed in the figure below:

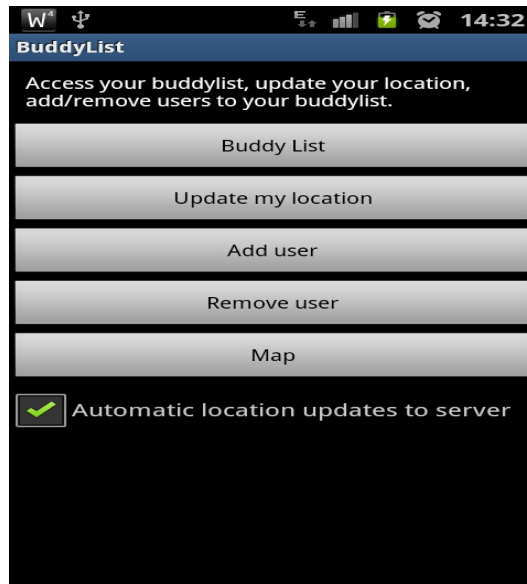


Figure 3-8: User interface, Buddy List client

In Figure 3-8 we have a number of buttons. Each of them is explained below:

- Buddy List: Navigates to the Buddy List.
- Update my location: Send a location update to the server manually.
- Add user: Add a user to the Buddy List and the SQLite database.
- Remove user: Removes a user from the Buddy List and the SQLite database.
- Map: show your current location in the map.
- Automatic location updated: Check for periodically automatic updates to the server if this is desirable.

The second interface of the Buddy List application is the Buddy List interface showed in the figure below:

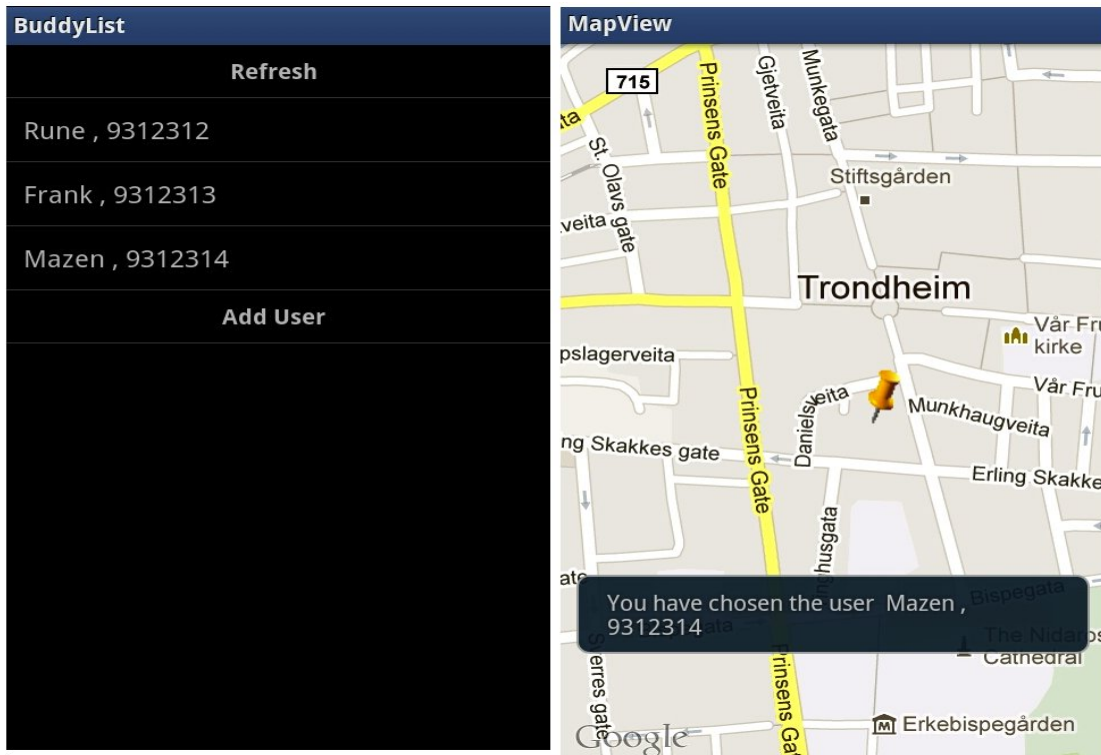


Figure 3-9: User interface, Buddy List and Map

In the left most screenshot of Figure 3-9 the users are represented with their name and phone number. If a user entry is pushed, a Google Map with their last updated location will be showed like the picture on the right side in Figure 3-9. If the refresh entry is pushed the Buddy List will request the Location Server for the users newest updated position and then update the database. In this section the Buddy List System developed in the previous project is briefly described. In the new system the Buddy List concept has been further developed and integrated into the system.

In this chapter the previous system for the service composition tool has been described. In the next chapter the new system for the service composition tool is presented.

4. System Overview, Design and Implementation

In Chapter 3 the previous system was presented. This chapter presents the new system developed during the Master Thesis period. The chapter is structured with a top-down approach. Hence, the reader will first get an introduction with an overview of the complete system and later get more detailed system description. The first section gives an overview of the system. The second section describes the client side of the system where a new GUI has been developed and the EasyDroid application has been re-made and developed further. Furthermore, the server side of the new system is presented in the fourth section. Finally the last section presents the interaction and communication between the different parts of the system. In addition the new functionality added to the tool is presented as well as some examples of the new version of the tool's usage.

4.1 System Overview

In this section a simple overview of the developed system is presented. The developed system consists of a client and server side with different components on each side, and the communication between them. The system overview is presented in the figure below:

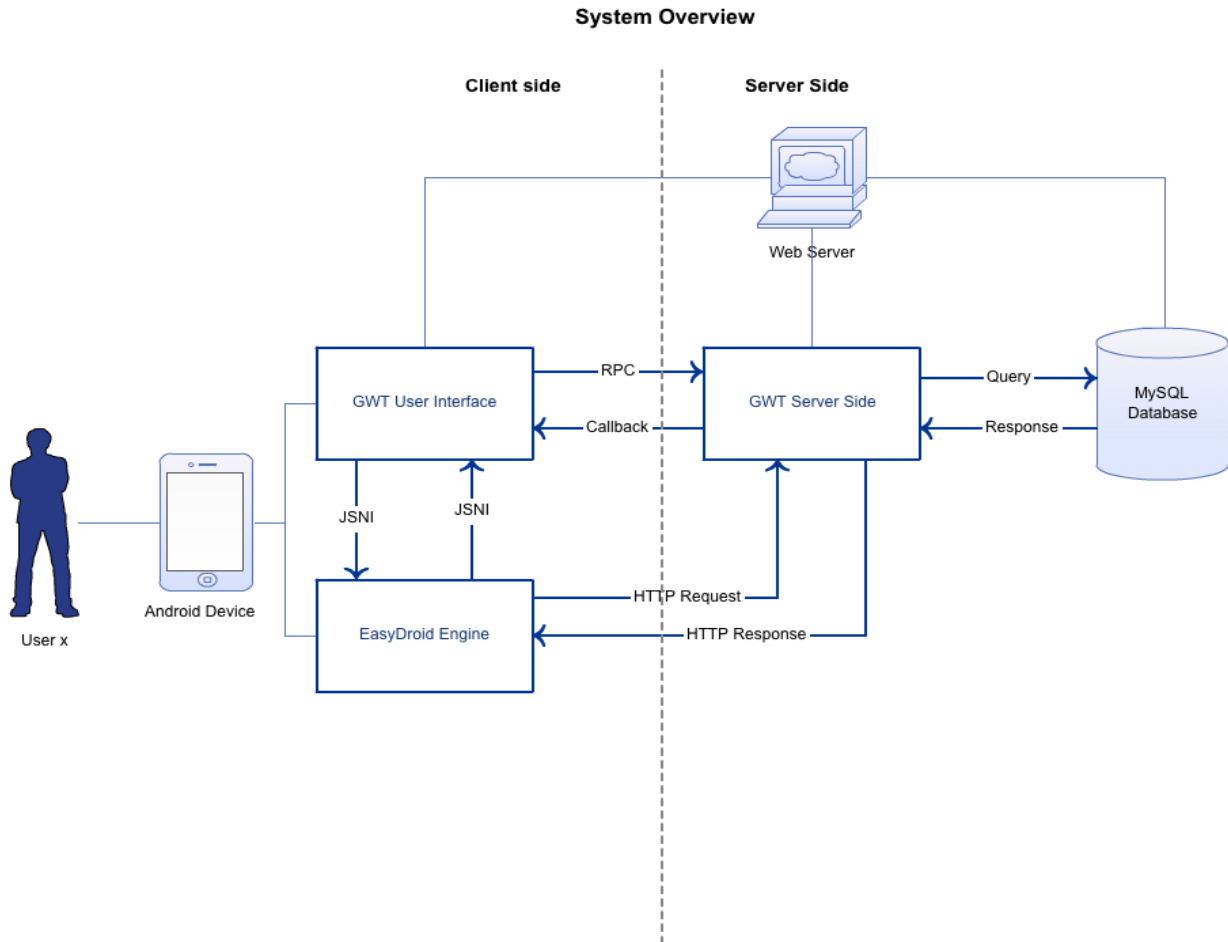


Figure 4-1: System Overview

In Figure 4-1, the system is divided into the client and server side. The client side contains a GUI developed and implemented using the Google Web Toolkit (GWT). The GWT GUI is a web application and therefore runs on a web server. Furthermore, the client side contains a service that runs on an Android device. This service is named the EasyDroid Engine. The GWT GUI is the equivalence to the existing Easy Composer application, and the EasyDroid Engine is the equivalence to the previous EasyDroid application. The same main principles apply for the new system. The GUI is where the user composes profiles, add locations, add calendar events etc. This information is uploaded to the database and downloaded to the EasyDroid Engine. The EasyDroid Engine runs constantly in the background and uses this information to activate profiles on either time or location.

The development of the Easy Composer tool was discontinued due to the fact that it had limited versatility in terms of platforms it could be accessed. In addition, the JavaFX script was discarded by Oracle in 2011 [25]. The new GWT GUI is web based, and the user can access the application as long as the user has a device that can access web sites. The EasyDroid Engine has almost the same functionality as the existing EasyDroid application. However, the previous Easy Composer has been replaced with the new GWT GUI and new functionality has been added. The previous EasyDroid engine and interface was strongly connected. Hence, we had to create a new engine almost from scratch using some of the classes and methods from the old engine. The new EasyDroid Engine should run in the background on the Android Device independently of the GWT GUI. The GWT GUI and the EasyDroid Engine communicate by using the JavaScript Native Interface (JSNI) that was described in section 2.4.2.

The server side consists of the GWT Server side code and the database. The GWT Server side functionality is primarily for making queries to the database, but has some other functionality as well. The GWT GUI communicates with the server side by making an asynchronous remote procedure call (RPC) with callback as described in section 2.4.1. As we can see from Figure 4-1, the GWT GUI client and server side resides on the same web server, which will make the remote procedure calls more rapid and efficient. In a scenario where the user does tasks on a device not running the EasyDroid Engine, the EasyDroid Engine needs to communicate with the server independently from the GWT functionality. The EasyDroid Engine needs to be synchronized with the server database in order to be updated with the latest changes. This type of synchronization is not implemented yet. In the future this synchronization will be achieved by making an HTTP request to the server and consequently receives the latest information from the database in the HTTP response. The next three sections will give an in depth description of the client and server side as well as the communication between.

4.2 Client Side

The Client Side consists of the GWT GUI and the EasyDroid Engine. The GWT GUI resides on a web server and the EasyDroid Engine runs on an Android device. The two first sections will give a presentation of the GWT GUI. Initially, the GUI visible to the user is presented, and then the implementation and architecture of the application is described. The third section describes the EasyDroid Engine of the client side.

4.2.1 Graphical User Interface (GUI)

In this section the methodology and result of the new GUI created for the tool is presented. Early in the project phase we started a process where we tested the existing tool on real-life users. Initially, a test plan was set up and a questionnaire [Appendix B] was made for user feedback. For a period of two weeks the users tested the tool, and at the end of the test period the users returned the questionnaire. Based on the feedback and our own experience we decided that we wanted a single common GUI instead of the existing tool's two interfaces. In order to have a common GUI for both desktops and mobile devices we decided that the GUI should be web-based. The implementation of a web-based GUI could be done in several ways. Finally, after doing some research and having discussed the possibilities the decision fell on using Google Web Toolkit (GWT) [18].

Before the actual implementation was carried out, a paper prototype testing of the GUI was carried out. Sketches of the layout with buttons, labels, windows, grids and more were made. Then a meeting was held where a user was testing the paper prototype and observers were taking notes. By creating a paper prototype we could detect design problems early and efficiently do changes based on the feedback from the test users. Eventually, a paper prototype was finished where the test users and developers were both satisfied. Finally, the implementation of the GUI could start.

The GUI was implemented using GWT and a GWT-based framework Smart GWT that allows for utilizing a comprehensive widget library. When designing and implementing the GUI it was important to apply common design principles and guidelines that were discussed in section 2.1.3 in order to achieve a user-friendly design. The rest of this section will present the different parts of the resulting GUI. At the end of each part where the GUI is presented a brief discussion about the applied design principles is presented.

4.2.1.1 The Log In

The Log In GUI is the first screen the user will see when starting up the application. The user types in a phone number and password, which is checked in the database. The user needs to be authenticated for security and privacy reasons. The Log In GUI is presented below:

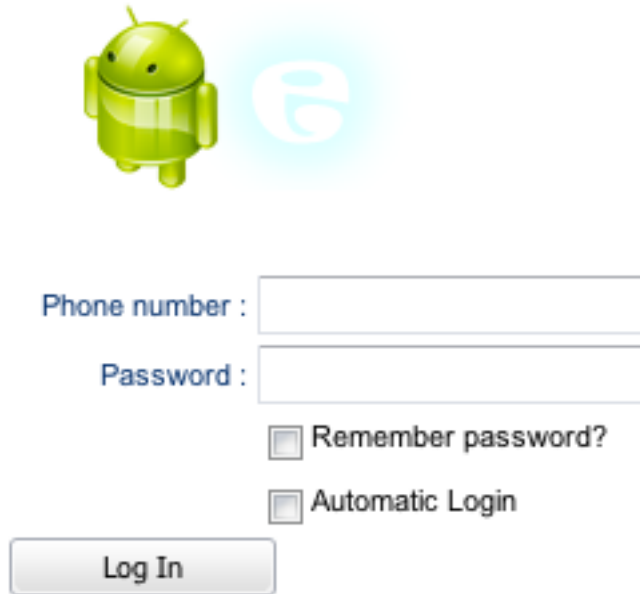
The image shows a login interface. At the top left is a green Android robot icon. To its right is a glowing blue circular logo containing a white stylized letter 'G'. Below these icons are two text input fields. The first is labeled 'Phone number :' and the second is labeled 'Password :'. Under the password field are two checkboxes: the first is labeled 'Remember password?' and the second is labeled 'Automatic Login'. At the bottom left is a rectangular button with rounded corners and a gradient, labeled 'Log In'.

Figure 4-2: Log In

The Log In GUI contains phone number and password text fields and a login button. In addition, the user can remember the password and also log in automatically by checking the correspondent check boxes. If the authentication is successful the application is loaded.

4.2.1.2 The Profiles

The composer part of the GUI was perhaps the trickiest part to implement and design. Since the GUI is supporting different platforms with different sizes and pointing devices, the composition area in the profiles feature needs to adapt to this. When the screen size gets below a certain threshold (smart phone screen size), the service composition tools traditional way of composing services gets useless. The solution was to make three different composition areas that are depending on the screen size. This resulted in a profile interface for large screen sizes (desktops, laptops), one for medium screen sizes (tabs) and one for small screen sizes (Smart Phone). Here the profile interface with the largest screen size is showed:

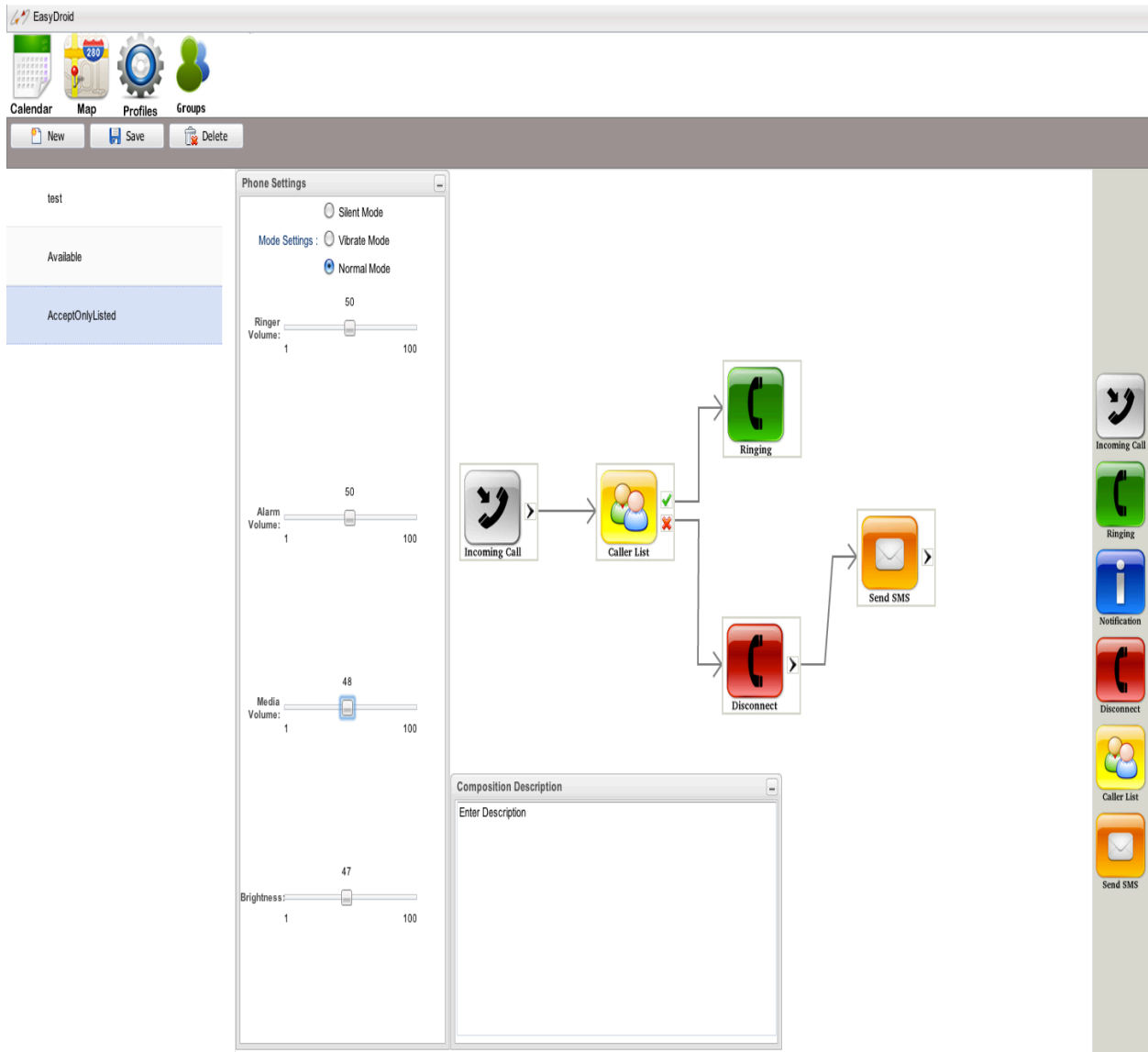


Figure 4-3: Profiles, Large size

In the top menu you can choose between three icons that lets you create a new composition, save the current composition or delete it. On the left menu, you can choose from already existing compositions. The main window is the composition area. On the left side of the composition area you can choose the appropriate phone settings for the composition you are creating in a Phone Settings window. There is also a window for describing the composition. These windows can be dragged and resized. Hence, the user can customize the position and size of these windows. On the right side there is a menu with six different components. Each of these components resembles a telephone service.

The six different components are:

- *Incoming Call*: Represents an incoming call.
- *Ringin*g: Connects the incoming call, making the phone starts ringing.
- *Notification*: When the Notification component is clicked, a window with a ticker, title and message field appears. This component notifies the user of actions that has happened in the background. For example if a call has been disconnected the user will get a notification of the event that has happened.
- *Disconnect*: Disconnects the incoming call.
- *Caller List*: When the Caller List component is clicked, a window with a list of numbers appears. Here you can add or remove numbers and the dropdown menu also lets you choose predefined groups of phone numbers. The list of numbers that can be used as a filter for incoming calls. For example the *Caller List* can either make a blacklist by having the numbers in the caller list disconnected, or a V.I.P list where only the numbers in the list are connected.
- *SendSMS*: When the SendSMS component is clicked a window with a phone number and message field appears. This component sends an SMS to the specified number. For instance in the event of a phone call is disconnected, the user can make use of the SendSMS component to send back a message to the disconnected user.

By adding and connecting these components, we can make a huge variety of different profiles. The connections are represented by arrows, which show the flow of service creation. The composition area resembles the composition area in the previous system, and the same control flow-based composition approach described in section 2.3 is used. In the figure above, a similar profile to the “AcceptOnlyListed” profile presented in section 3.1.1 is created.

When the application is used with Tab devices, the Profiles feature presented above becomes too large. The Profile feature for medium screen sizes is presented below:

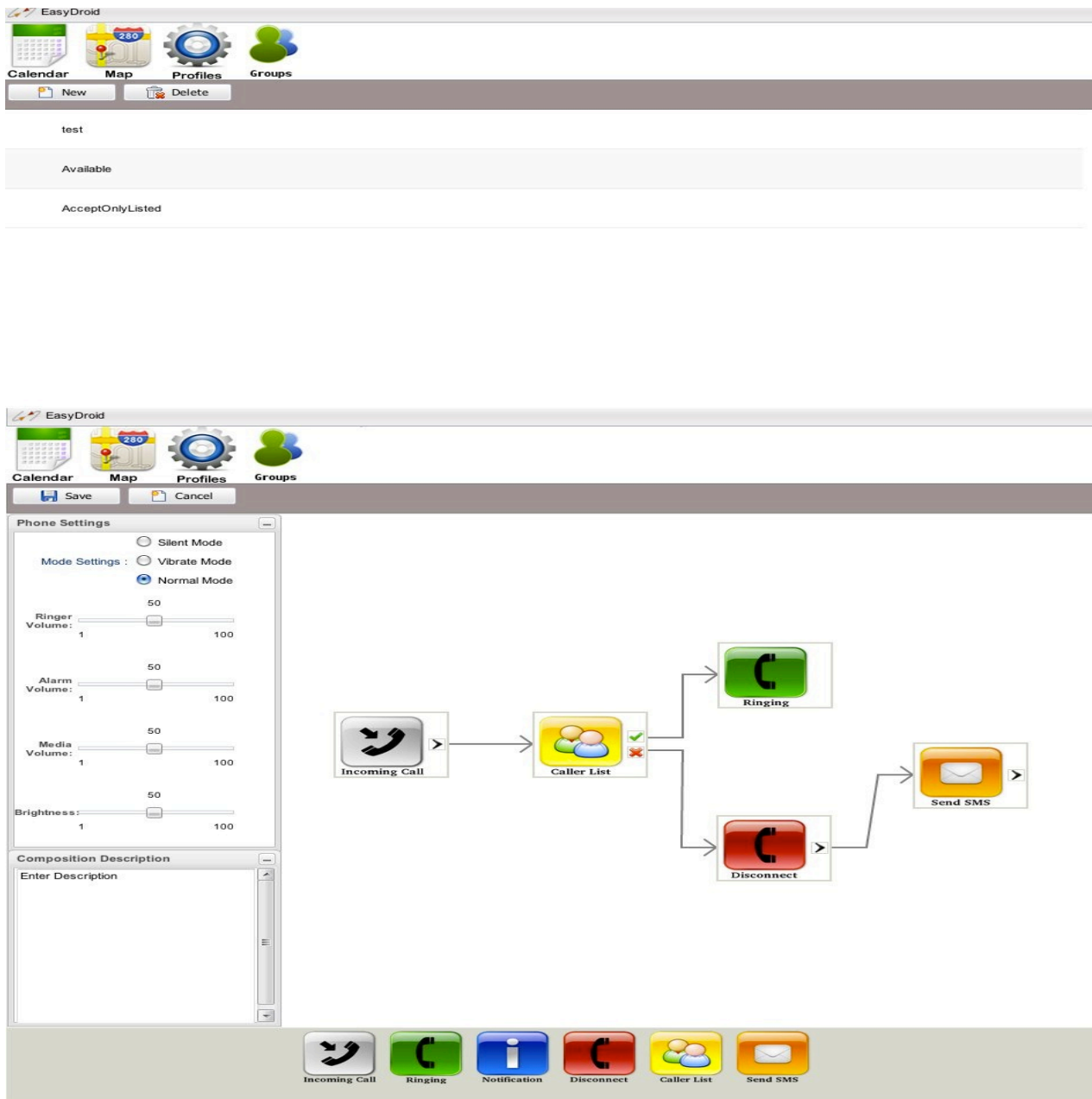


Figure 4-4: Profiles, Medium size

For the medium sized screen version of the profiles the user is first presented with a list of profiles as showed in the top screenshot of the figure above. The user can choose between creating a new profile and editing an existing one. In the second screenshot of Figure 4-4, the user has chosen the AcceptOnlyListed profile, which navigates to a new screen consisting of the composition area. If the user had chosen to create a new profile the composition area would be empty. Basically, the difference between the large size and medium size Profile GUI is that the medium size has divided the profile list and composition area into two separate screens to save space. Furthermore, the components bar has changed to being horizontal at the bottom instead of vertical on the right side. Hence, the composing of the profiles works in a vertical way, moving will create more space vertically. The other functionality remains the same.

The small screen version of the Profiles functionality features a completely new way of composing profiles. The drag and drop component way of composing profiles of the medium and large screen size versions require a certain screen size, and doing this on a small screen is cumbersome. In the small screen version, the profiles are composed by making use of tree-structured columns. The following screenshots illustrates how the composing is carried out:

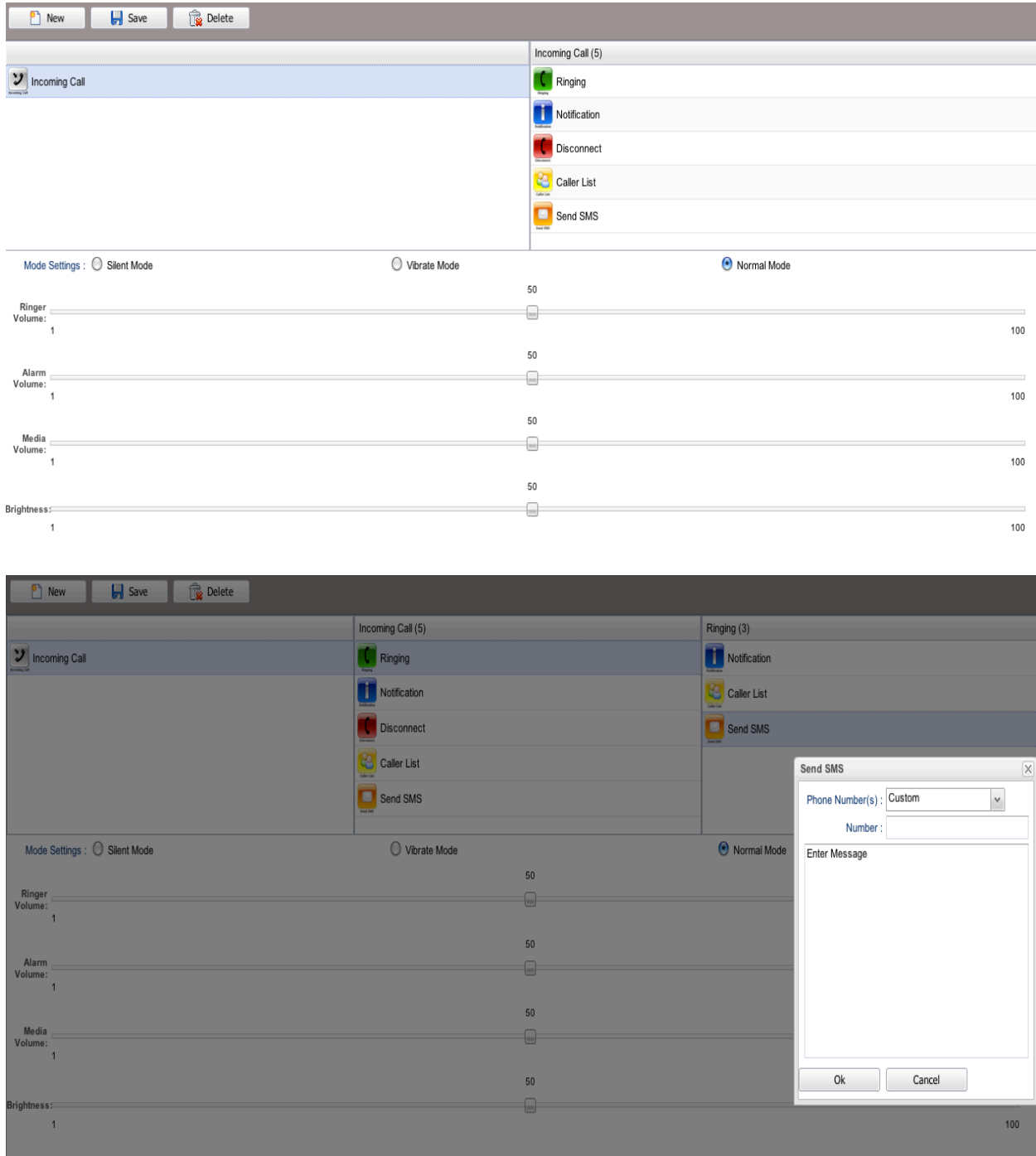


Figure 4-5: Profiles, Small size

When a component is clicked, a new column with new components appears. This is illustrated in Figure 4-5 by first clicking the Incoming Call component and then clicking the Ringing compo-

ment. This process repeats itself until you reach a leaf node of the tree. If a SendSMS, Notification or Caller List component is clicked, a window pops up where you can type in the relevant data. In Figure 4-5, the SendSMS component is clicked. Beneath the composition area the user can set the phone settings. The menu bar on the top works in the same way as for the larger screen versions.

For designing the Profiles interface, a number of design principles have been used. The design principles mostly focus on the eight golden rules described in section 2.1.3. First of all, it is designed for portability so that the Profile GUI adapts to different screen sizes and pointing devices. Consistency has been assured with having the same component icons for the three different screen sizes. The action sequence of composing the service is also consistent for the large and medium sized display versions. Informative feedback is also assured by using simple graphical animation when dragging a component in the composition area. Furthermore, the completion of a sequence of actions gives informative feedback by giving a prompt when a profile is save. The composing of services also prevents errors, by having implemented a conflict resolution. In this way the user cannot make serious errors. The Profiles functionality also permits easy reversal of actions, by allowing the user to delete components and arrows in the composition area and also editing and removing already created profiles. Finally, the navigation requires no short-term memory load on the user.

4.2.1.3 The Map

The new Map functionality of the application has been greatly improved from the previous system in terms of usability and utility. The Map GUI is the same for all screen sizes. The figure below shows the Map GUI with a location added:

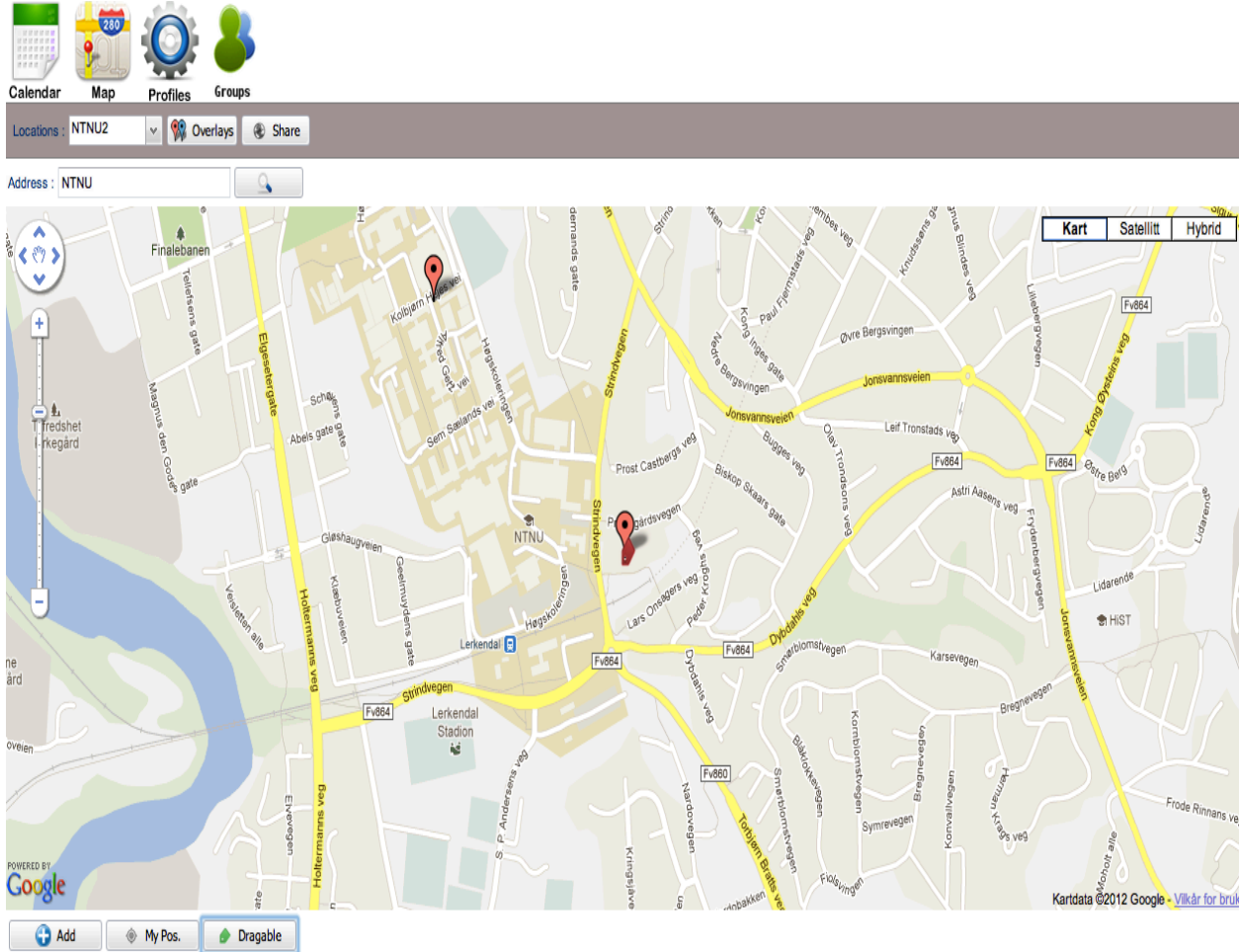


Figure 4-6: Maps with location marker

The Map GUI consists of a top and bottom menu of buttons. In the middle there is an address search feature and a map area. The map area is powered by Google Maps, where you can navigate in all directions, zoom in and out and choose between satellite, map or a hybrid view. The bottom menu buttons are simpler functionality directly related to the Map while the top menu buttons offers more advanced functionality. For adding a location the user has a choice between searching for the address by using the search functionality or simply using a *dragable* marker that you can drag around the map. The search functionality is showed in Figure 4-6 as a search field and a search button located in the bar directly beneath the menu bar. The *dragable* marker is showed in the middle of the map, and is activated by pressing the button to the bottom right. When the *dragable* marker is activated, the button turns green as showed in the figure.

In the bottom menu of the Map GUI, the most left button is for adding a location from the address or where the *dragable* marker is located. Pushing this button results in a popup window with a location name text field, and the option to save it or cancel. When an address search is done the map will be centred on the coordinates of the location and the *dragable* marker functionality will be activated with the *dragable* marker centred on the location. If the user choose to add this location a marker is showed where the address is and the location is saved. In Figure 4-6 we have done a search for NTNU and added a location for this address search result. When the *dragable* marker functionality is activated a marker will be placed where the *dragable* marker is located.

In the previous system, it was only possible to search for known addresses when adding a new location. This can be cumbersome if the user has no knowledge of the exact address or if the location does not have an address. The solution for this was adding the *dragable* marker functionality. The user can navigate the marker within the map, until the marker is on the correct spot. By implementing this the user has the options of searching for an address he might not know the location of, or adding a location that he does not know the address of.

The button in the middle of the three in the bottom part of the map interface is for showing your own location when the application is running on an Android device. If it is activated, the button is highlighted in green. Getting to know your exact location could be useful for adding a location nearby and also help the user decide if the user location information should be shared with others. This is a part of a new functionality for the tool called the Buddy List System. The Buddy List System is a location-aware system that sends user location updates to a server, and can also receive other users location updates from the server. The will be explained in greater detail later in this chapter.

When a location is added, it is stored and showed in a dropdown table as showed in the figure below:

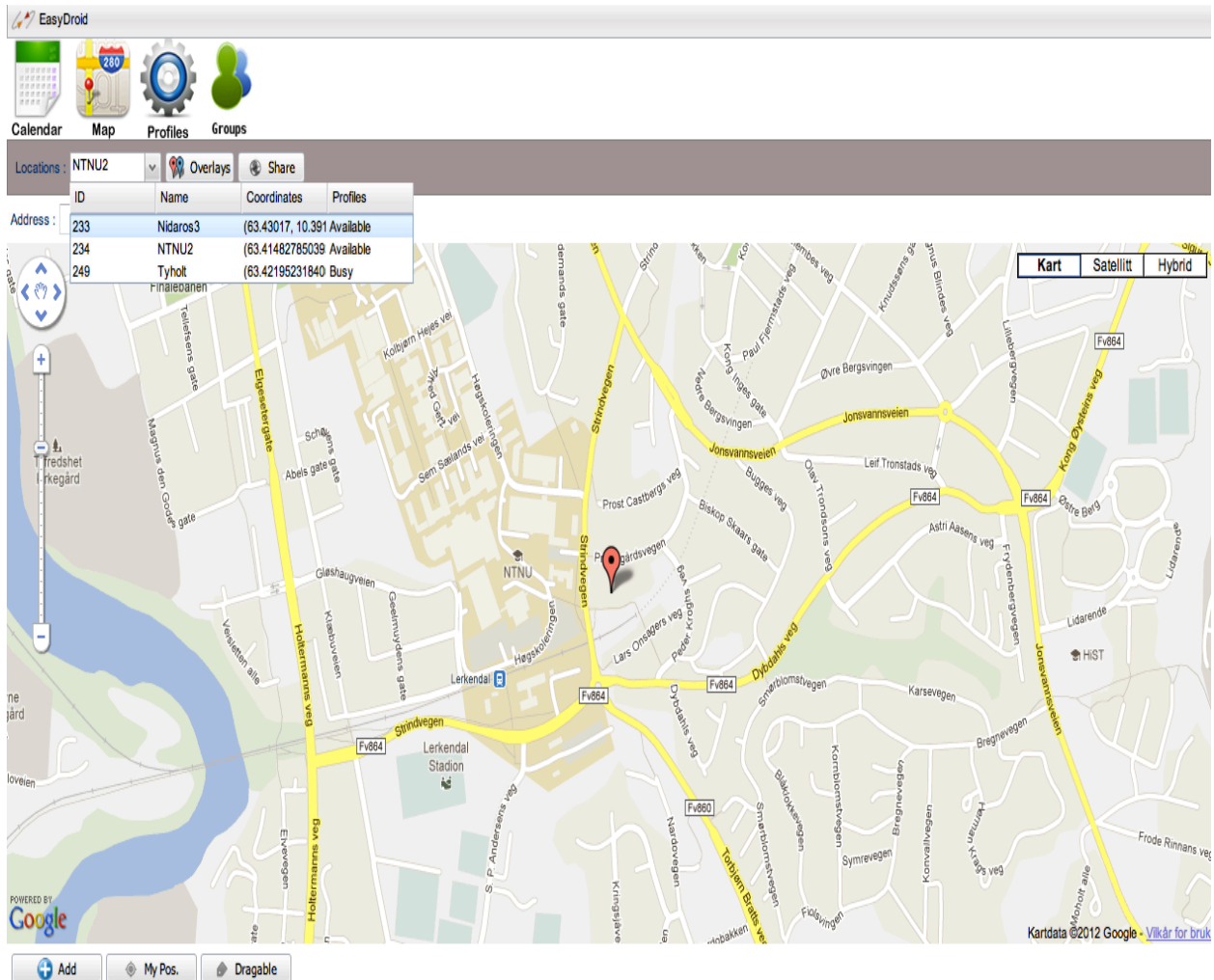


Figure 4-7: Maps GUI, with location table

In the dropdown table there are three records with the fields Id, Name, Coordinates and Profile. When a record is chosen the map automatically navigates to this location. When a pointing device hovers over the marker, information with the location name is presented. For adding or editing the profile on a location the user pushes the location marker on the map. When the location marker is pushed a location settings window pops up. This is showed in the following figure:

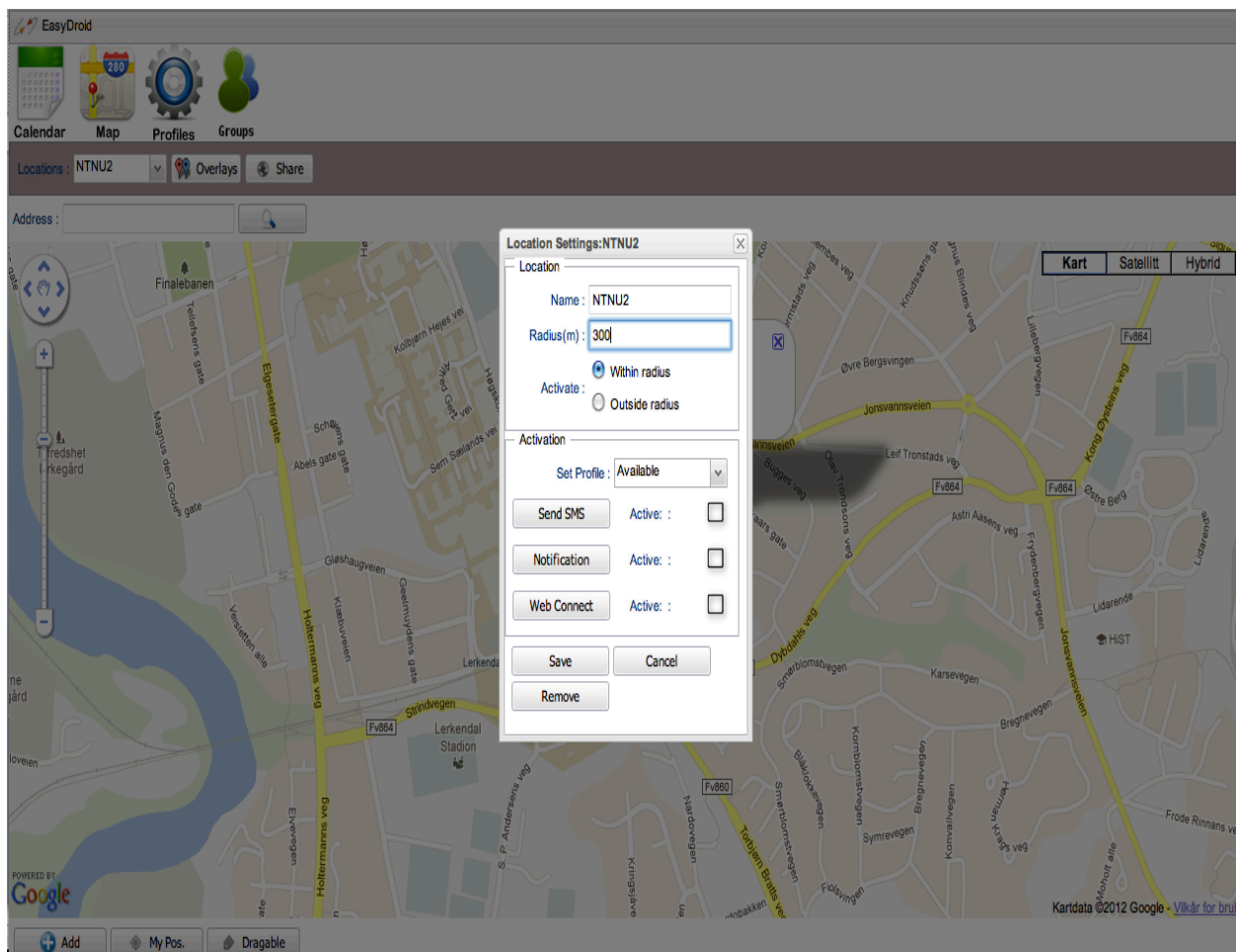


Figure 4-8: Map, location settings window

In the location settings window you can change the name of your location in the name field. The radius field and the activate toggle buttons is where you choose if the profile should be activated within or outside the radius and the number of meters of the radius. The radius field only accepts integers. Hence, the user gets feedback showing that the input is wrong if he types in other characters than integers. This is important for preventing the user making any input errors.

The Activation section is where the user chooses what should be activated based on the fields above. Similar to the existing tool, a profile can be activated for the location. In the figure above the chosen profile is the “AcceptOnlyListed”, which was made earlier in this section. Furthermore, for the location settings we introduce a new functionality called “Fast Activation”.

The Fast Activation consists of the Send SMS, Notification and Web Connect features. In the check boxes to the right of the Fast Activation buttons the user can choose whether each feature should be active or not. They can all be activated at the same time. When pushing the Send SMS button, a window pops up that allows the user to write a text message and which number to send it to. For the Notification the user can write a message, title and ticker. In the Web Connect the user can set a URL. The basic idea of the Fast Activation is that the user can send a text message, get a notification or connect to a pre-determined web site when it reaches a location. The Fast Activation concept will be described in detail later in this chapter. Finally, the user can save the

location settings or cancel it by pressing the Save or Cancel buttons. The user can also remove the location by pressing the remove button.

Another improvement from the previous system is that the user can set which overlays to be showed in the map. By pressing the Overlay button right next to the location dropdown menu an Overlays settings window pops up. This is showed in the figure below:

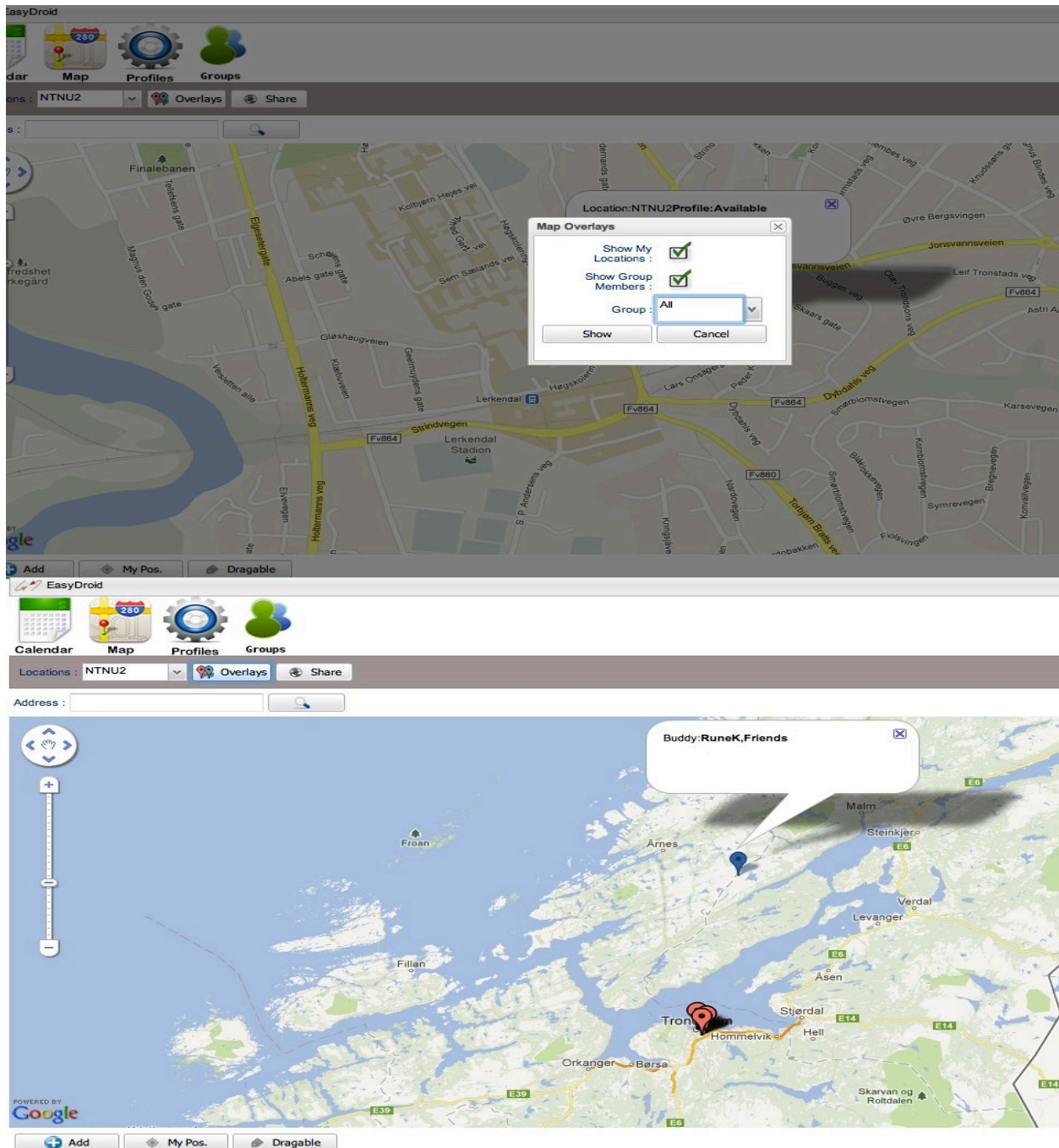


Figure 4-9: Map, Overlays

In the Overlays settings the user can set which overlays that shall be showed in the Map by checking the two check boxes. This was not possible in the previous system. Furthermore, this feature lets you show where other users of the tool are located. These users are stored in the Groups functionality, which will be described later in this section. The user can show a specific group of members or all the members in Groups by choosing from the Group drop-down menu. Depending on what the user has checked, markers are added or removed when the show button is pushed. In the bottom screen of Figure 4-9, the markers are added. The red markers are static locations, and the blue markers are members of Groups. When a pointing devices hovers over the markers, information about the location or member is displayed. The Groups function a part of the previously mentioned Buddy List System, which will be described later in this chapter.

Finally, the button located to the right in the top menu of the Map interface is directly connected to the Buddy List System. When pushing this button a window with location sharing settings appears. The figure below shows the location sharing settings:

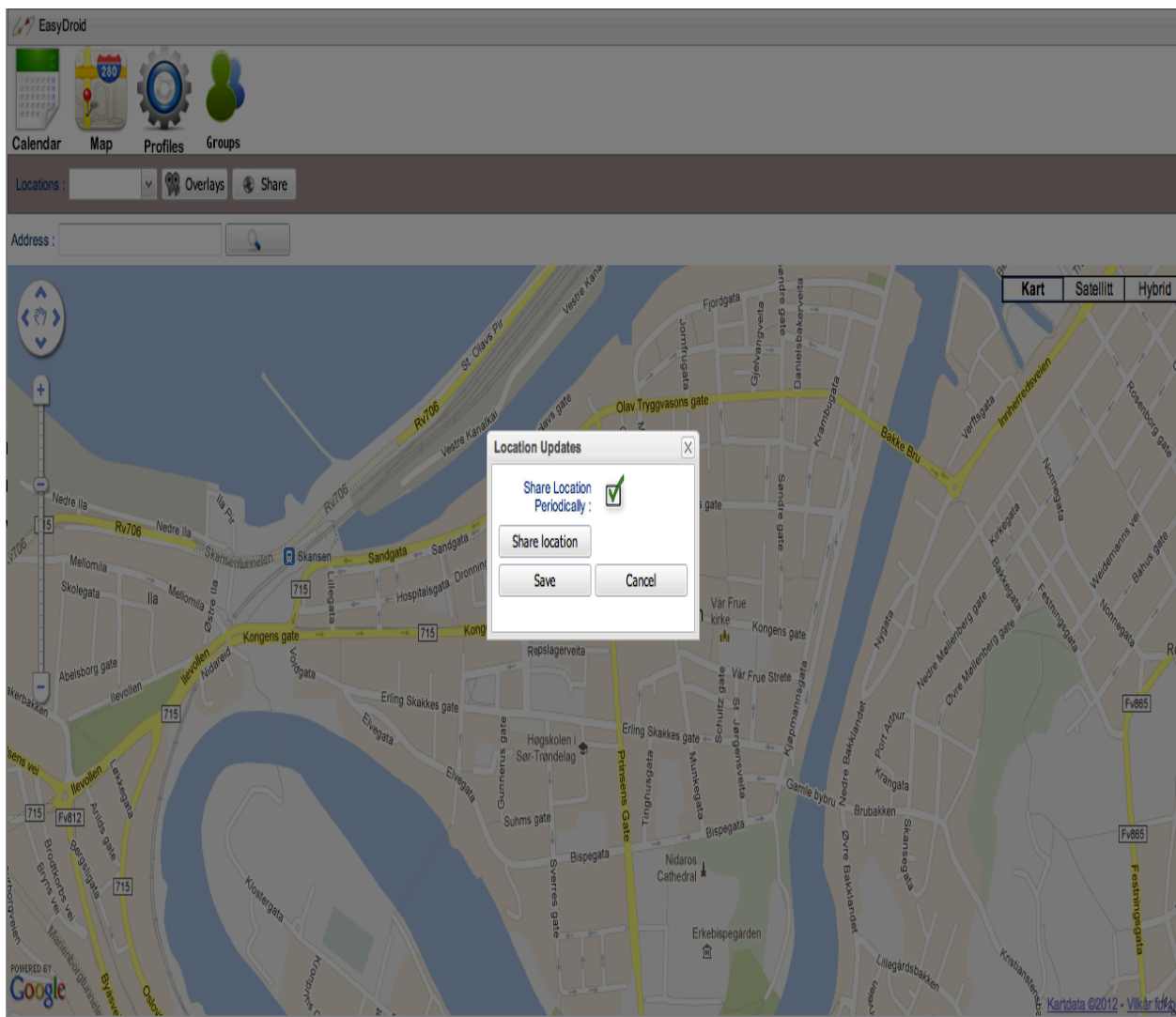


Figure 4-10: Map, Location sharing

The location sharing settings let the user determine if the user location should be shared periodically by checking the checkbox. If the user wants to share his location manually, he can press the share location button when it is desirable and leave the checkbox unchecked. The location updates is sent to a server and is a part of the Buddy List System.

In the Map GUI the buttons are highlighted if their functionality is active. The highlighting is a type of feedback to the user. The pop-up windows are also consistent in the sequence of action and their layout with buttons, fonts and colours. When a location is added, or a profile is added to a location, the Map GUI gives feedback by showing the new marker on the map or giving a prompt when a profile is saved. The Map GUI also prevents errors by making the user location button inaccessible when using the application on a device that is not running the Android OS. Checking if the input of the radius is an integer also prevents input errors. The Map GUI also permits for easy reversal of actions, by allowing for deleting and editing the locations. Finally, it reduces the short-term memory load by saving the state of the location settings so that it is consistent for closing and opening the location settings window.

4.2.1.4 The Calendar

The new Calendar functionality is similar to the functionality in the existing tool. It has two different interfaces, which are dependent on the screen size. We wanted to keep the existing layout for the application, since it had adequate quality in terms of usability. The following figure shows the Calendar GUI:

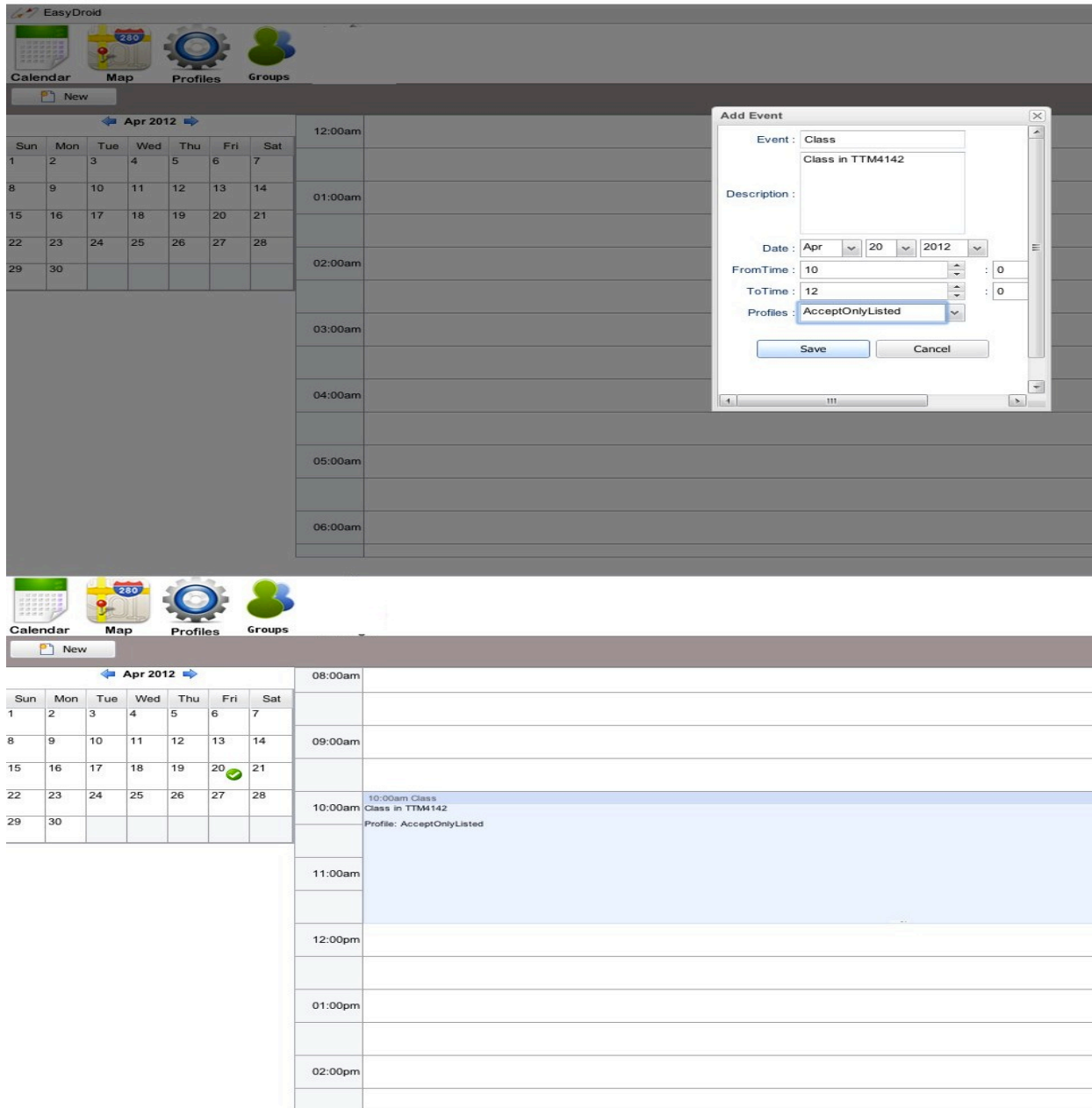


Figure 4-11: Calendar interface, Adding a time event

Figure 4-11 shows two screenshots of the Calendar GUI. To the left in the GUI, there is a small calendar where the user can choose a specific date. When a specified date is chosen, a corresponding time schedule appears to the right. When the user wants to add a new event to the cal-

endar, the “New” button is pressed and “Add event” window like the one in the first screen shot appears. In this window the user can determine the name and a description of the event using text fields. For choosing the date and time period the user must choose from predefined values in dropdown menus and increment/decrement fields. To have this predefined is important for preventing the user of making any input errors. Finally, the user can determine which profile that should be activated for the time event. If the event is saved, it is showed in the time schedule for the specific date as showed in the second screenshot of Figure 4-11. The user can edit the event by simply clicking on the event in the time schedule. Furthermore, an added event is also showed in the calendar by a small green icon as in the second screenshot of Figure 4-11. The next figure presents two screenshots of the compact Calendar GUI:

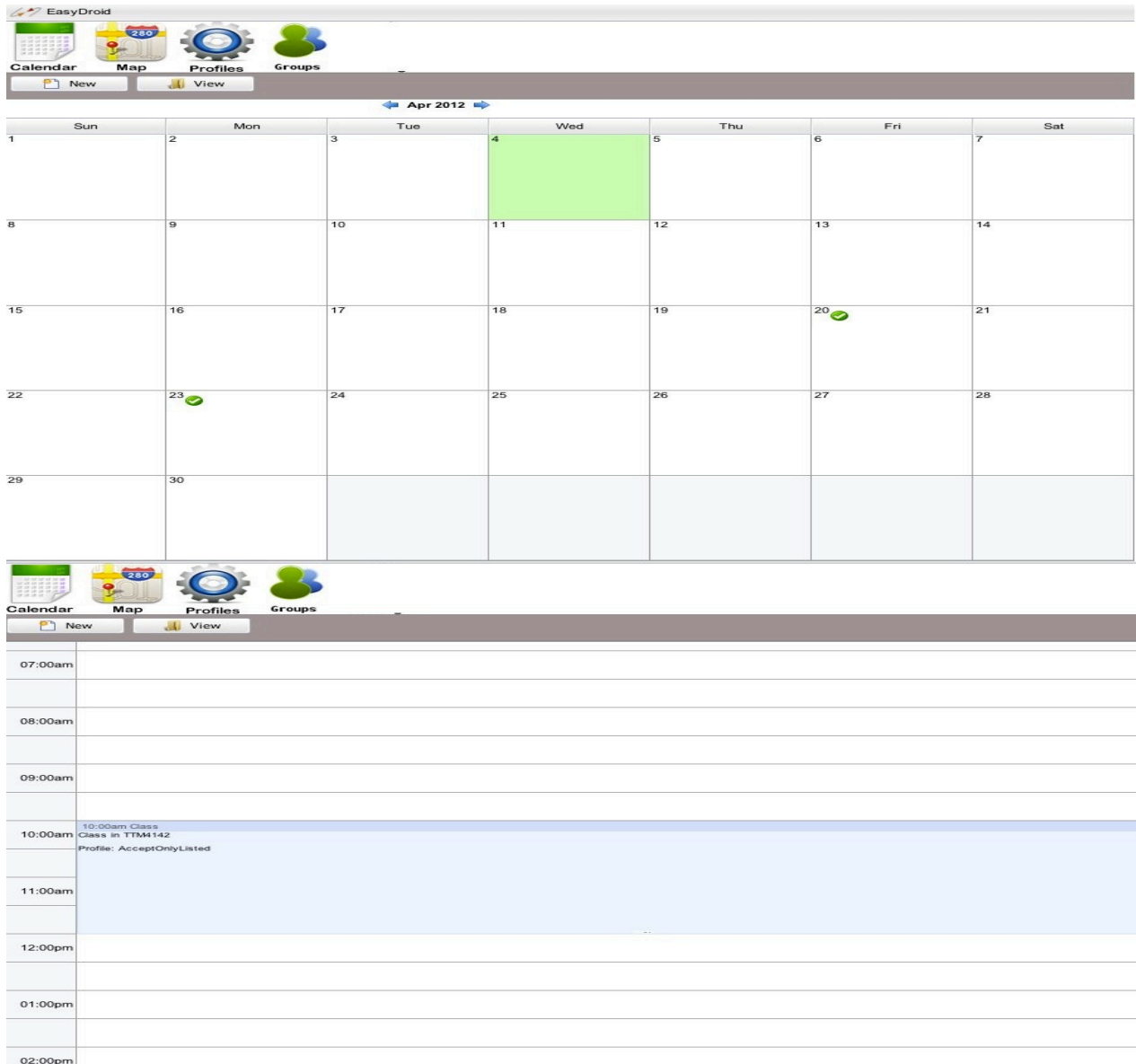


Figure 4-12: Calendar, Compact GUI

In the compact version of the Calendar GUI the user is first presented with a calendar consisting of large squares as showed in the first screenshot of Figure 4-12. This will make it easier for the

end user to choose the desired date than using the small calendar for the large screen version, which would be even smaller on a compact screen. The dates are highlighted in green when a pointing device hovers over them. When a date is chosen the Calendar navigates to a new screen with the time schedule for the specified date. The Profiles principle of dividing functionality into two screens to save space is also used here. Otherwise, the functionality of the compact Calendar GUI is the same as for the larger screen version.

The Calendar GUI allows for portability by having different screen sizes for different platforms. Also, the different screen size versions are consistent in terms of action sequences, buttons, icons, text etc. The GUI also offers informative feedback by showing that an event has been added by highlighting dates where events has been added and also give a prompt that the information has been stored in the database. Having dropdown menus with pre-determined values prevents input errors, such that the user can't make serious errors by typing in wrong characters for date and time. The GUI also permits easy reversal of actions by allowing deleting and editing the events in the time schedule.

4.2.1.5 The Groups

The Groups functionality in the new system has drastically improved in terms of usability and utility. In the previous system, the Groups functionality was limited and in an early phase of development. The new Groups functionality has become more flexible and has some neat new features added to it. The Groups GUI is presented in the figure below:

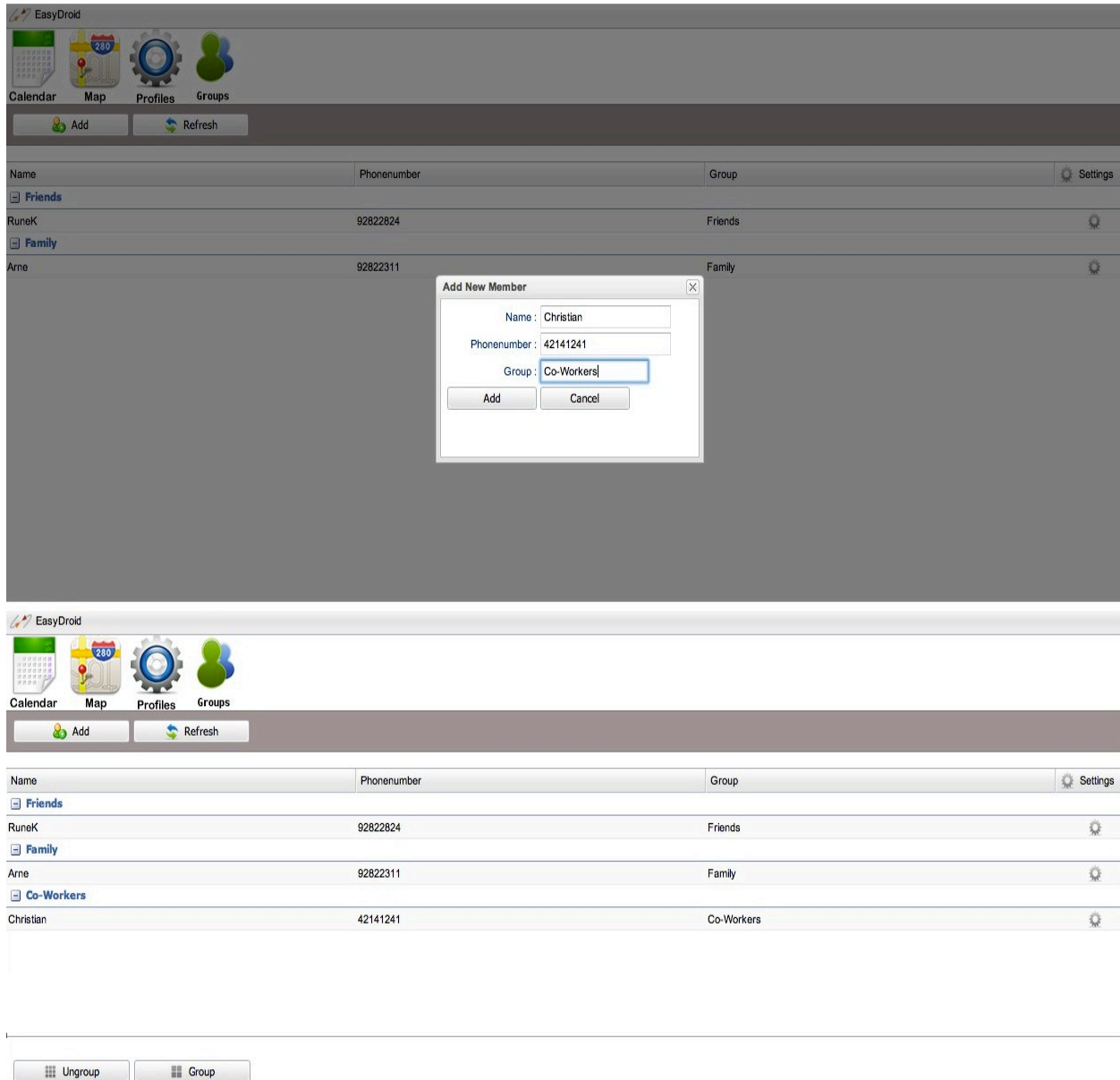


Figure 4-13: Groups GUI, Adding a Member

As showed in Figure 4-13, the new Groups GUI consists of a top menu bar, the Groups table and two buttons located at the bottom. The first button to the left in the top menu bar is for adding a member to the Groups table. When this button is pushed, the *Add Member* window similar to the first screenshot in Figure 4-13 appears. The *Add Member* window contains name, number and

group text fields. When the member is added, the Groups table is updated. If the group text field contains a group that does not exist in the Groups table, a new Group is automatically created. Furthermore, the new Groups functionality is more flexible as it allows for the user to edit the fields of already added member. If the user changes a members group, this is automatically updated in the Groups table. The user also has the option of choosing which groups that are visible by expanding and collapsing the groups. The two buttons at the bottom is for grouping and ungrouping the group members in the table. The ungrouping button is the one to the left. When the ungrouping button is pushed, the Groups table becomes plain list with group members. This could come in handy if the user has a large group list of members and wants to find a user. The user can then order the ungrouped list alphabetically by pressing the *name* column at the top of the Groups table. The *onenumber* and *groups* columns also have the ordering capability. For grouping the members again the user can push the button located to the right.

From Figure 4-13 we can see that the Groups table rows consists of three editable text fields and a *settings* button. The *settings* feature allows the user to determine the settings of each user. In the new Groups functionality the members of the groups can also have a location attached to it. The figure below shows what happens when the *settings* button is clicked:

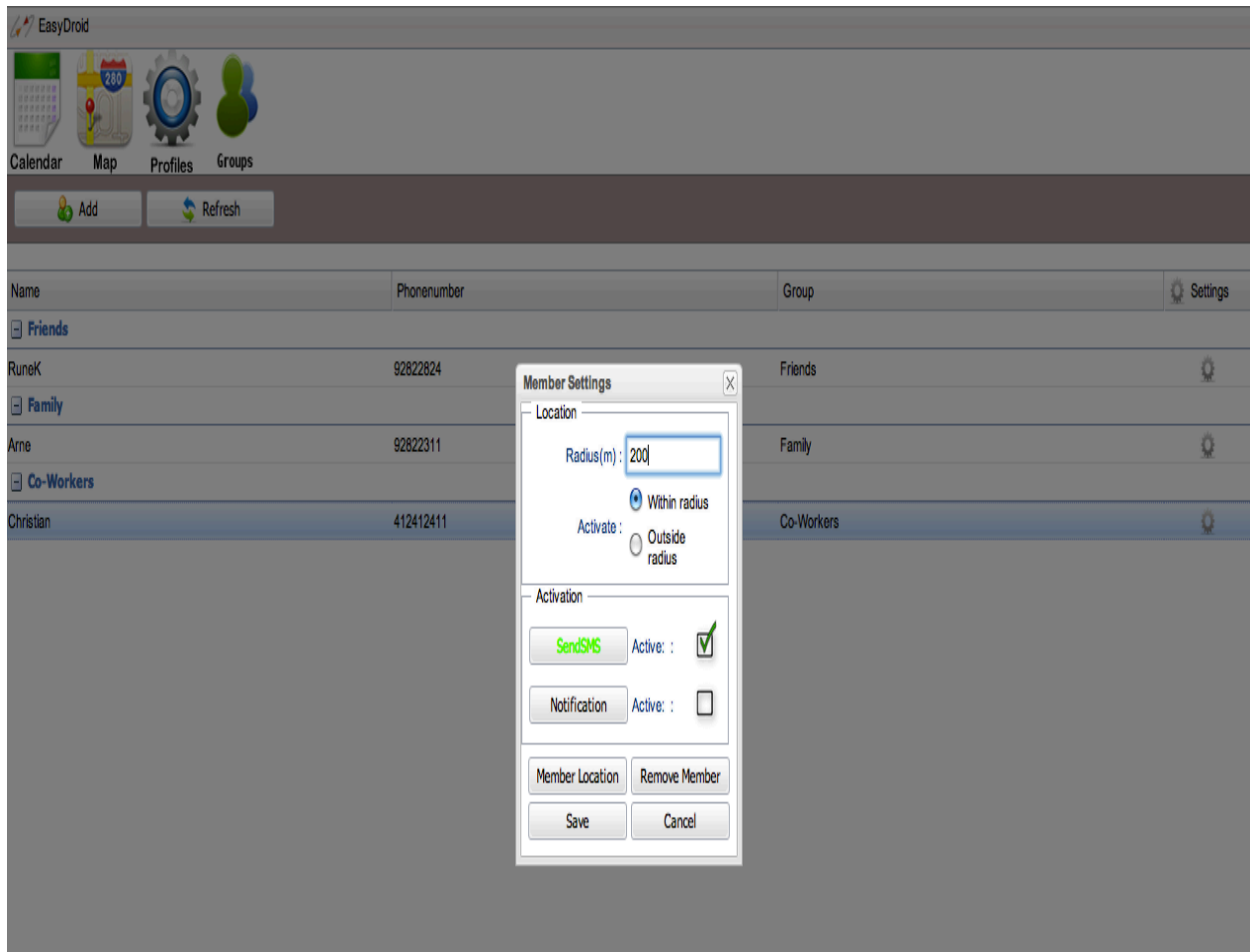


Figure 4-14: Groups GUI, Member settings

In the member settings window the user can display a single members location in a Google Maps window by pressing the *Member Location* button. This is a part of the previously mentioned

Buddy List System and is somewhat similar to the Maps Overlays feature that allows the user to see multiple users of one group in the same map. The user can also remove a member from the grid by pressing the *Remove Member* button.

Perhaps the most interesting feature with having a location attached to each member is allowing for the user to activate the *Send SMS* and *Notification* features when a member is within or outside the specified radius of the users location. This functionality is called the Member Activation. The Member Activation is similar to the Fast Activation feature for the Maps functionality, except for the Web Connect feature. Another important difference is that the Groups member locations change dynamically instead of being stationary. In other words, this new member settings feature is a combination of the new concepts Buddy List System and Fast Activation. Both of these concepts and the practical usage will be described later in this chapter.

One of the neatest features of the Groups GUI is its flexibility of user control. This means that the user can get the information in the most convenient way by ungrouping and grouping the rows and sorting the rows based on columns. The GUI is also consistent in the action sequences of tasks, buttons, icons, etc. Furthermore, informative feedback is given when updating the Groups table with a prompt that the database has been updated. Checking the input of the phone number and radius of the member settings prevents input errors. Allowing the user to edit and delete the members in the Groups table permits for easy reversal of actions.

The design of the GUI has been carried out with carefully planning in order to get the sufficient quality in terms of usability and utility. Before the design of the new GUI was initiated, the user test feedback from the previous system was used to make a proper requirement analysis in order to understand what features that could be improved. We learned that having two different GUIs created a longer learning time for the individual users, and having one interface would decrease the learning time. Furthermore, the interface would now have the same functionality for both the desktop version and mobile versions. This was not the case in the previous system. This means that the users are able to compose their own profiles on the fly, add locations, add profiles to the locations, add events in the calendar etc. All of which was unavailable for the mobile platform before. In addition, several new features and functionality has been added. Another argument for making a web-based GUI is that it would make versions for other smart phones such as Iphone and WinPhone easier to develop.

After the requirements analysis was done, a paper prototype test was done to make sure the design of the GUI was on the right track. The resulting GUI is applying the eight principles of interface design called the “golden rules” as well as other principles. The application is consistent by having similar action sequences in similar conditions, identical terminology in menus and prompts, and consistent use of icons, fonts, labels, names, colours etc. Informative feedback and error prevention are also emphasized principles. The application also supports easy reversal of actions, internal locus of control and reduced short-term memory load. How the principles are applied is discussed at the end of each functionality section presented previously. By applying these principles we know that the GUI will have a certain level of quality. However, in the end, what counts is the subjective user satisfaction of the GUI. This needs to be tested on real life users. The tests and results are presented in chapter 5. The next section will describe the client GUI in greater detail, presenting the implementation and architecture.

4.2.2 Architecture & Implementation of the GUI

In this section the architecture and implementation of the GWT GUI is presented in greater detail. The previous section only presented the surface visible to the user. Here the different classes and functionality features that lie behind the GWT GUI are described. The overview class diagram of the GWT GUI is presented below:

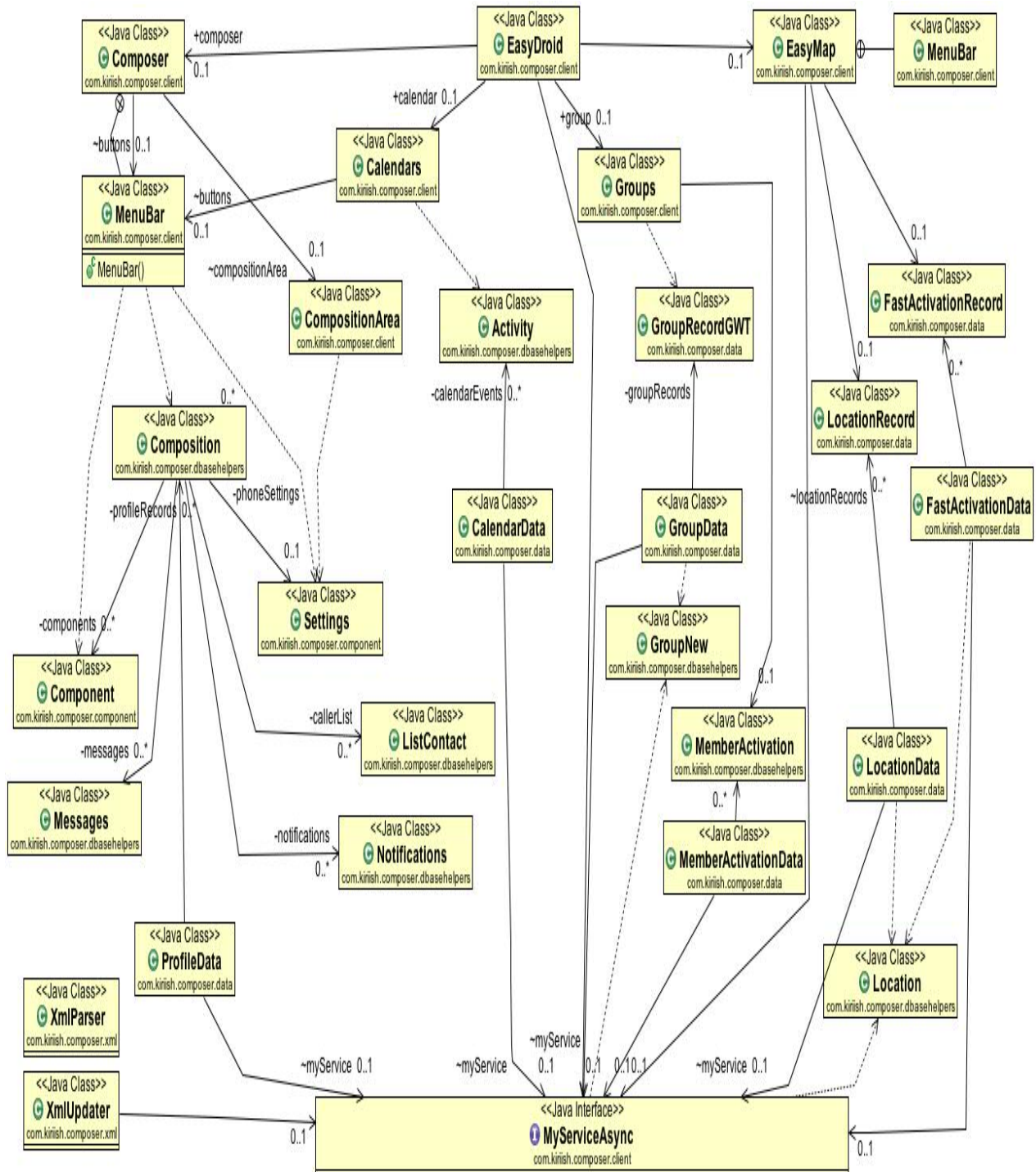


Figure 4-15: Class Diagram, GWT GUI overview

The class diagram in Figure 4-15 above shows an overview of the main classes of the client side GWT GUI. Some of the classes are left out of the diagram. These are classes with similar functionality or of minor importance. For instance the classes for the smaller screen versions of the composer is left out. Furthermore, the attributes and methods of the classes are also left out. The purpose of the overview class diagram is to get an overview of the architecture before looking deeper into the more detailed class diagrams. As we can see from the diagram the *EasyDroid* class is the main start-up class. The other main classes *Composer*, *Calendars*, *Groups* and *EasyMap* are initialized in the *EasyDroid* class. By looking at the GUI presented in section 4.2.1.2 we can see that each of these classes have their own icon in the top menu bar of the application. Dependent on the icon, one of these classes is instantiated when clicked. The next section describes the start-up functionality.

4.2.2.1 The Start-Up Functionality

This section describes the start-up functionality. The main classes are showed in the class diagram in Figure 4-16 and is followed by a description:

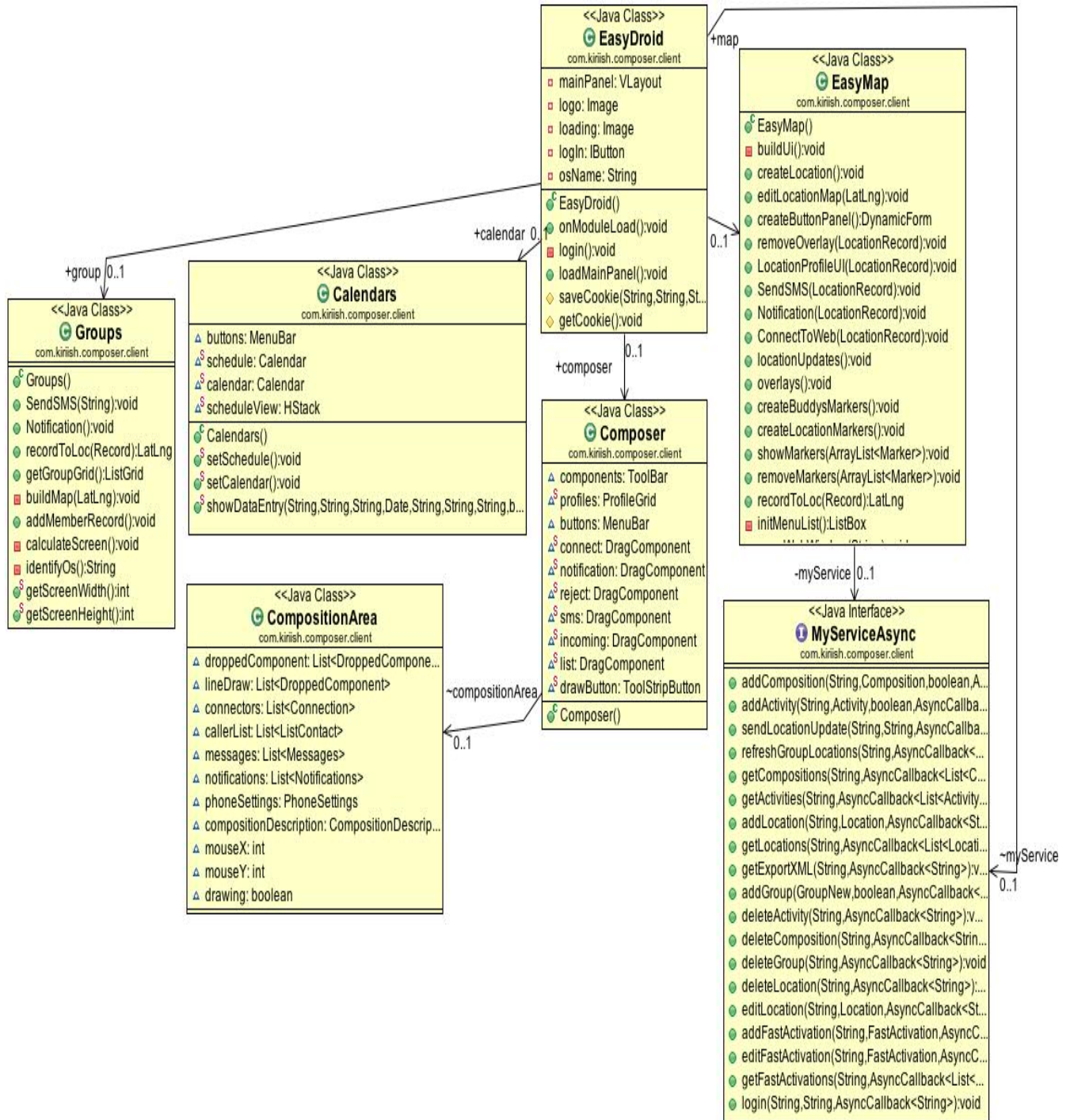


Figure 4-16: Class Diagram, Start up functionality

A description of the class diagram is presented below:

- The *EasyDroid* class is the start-up class where the other main classes are instantiated from. Initially, the class shows the Log In GUI in Figure 4-2. When the user clicks the *login* button, the *login()* method is called. The *login()* method calls the *login()* method from *MyServiceAsync* class which checks the user credentials in the database. If the authentication is successful the user number is set to this phone number. If the check boxes in Figure 4-2 are checked, this is saved in a cookie by the *saveCookie()* method, and retrieved by the *getCookie()*.
- After the log in, the *EasyDroid* class then calls methods to receive previously stored information from the database from last time the application was used with the phone number set.
- These methods will then populate the *EasyMap*, *Calendars*, *Composer*, and *Groups* data sources with the relevant data. The *EasyMap*'s data source will be populated with locations and location properties, the Calendar data source will be populated with events and profiles attached to events, the Composition data source will be populated with profiles and the *Groups* data source will be populated with members and members properties.
- The *EasyDroid* class also checks which OS the application is running on and the screen size of the device in order to set the appropriate pointing device and size for the GUI.
- Finally, if the application is running on an Android device, the *XMLUpdater* class methods *getXMLFromDatabase()*, *setFastActivationToAndroid()* and *setMemberActivationToAndroid()* are called. These methods call the correspondent methods from *MyServiceAsync* class, which pulls the Compositions, Fast Activation and Member Activation XML from the database and pass it to the *EasyDroid* Engine calling the JSNI methods *setXML()*, *setFastActivation()* and *setMemberActivation()* inside the *XMLUpdater* class.

This section has described the start-up functionality. The next section describes the Profiles functionality.

4.2.2.2 The Profiles Functionality

This section presents the Profiles functionality architecture and implementation. Below, Figure 4-17 shows the classes related to the Profiles functionality in a class diagram, followed by a description:

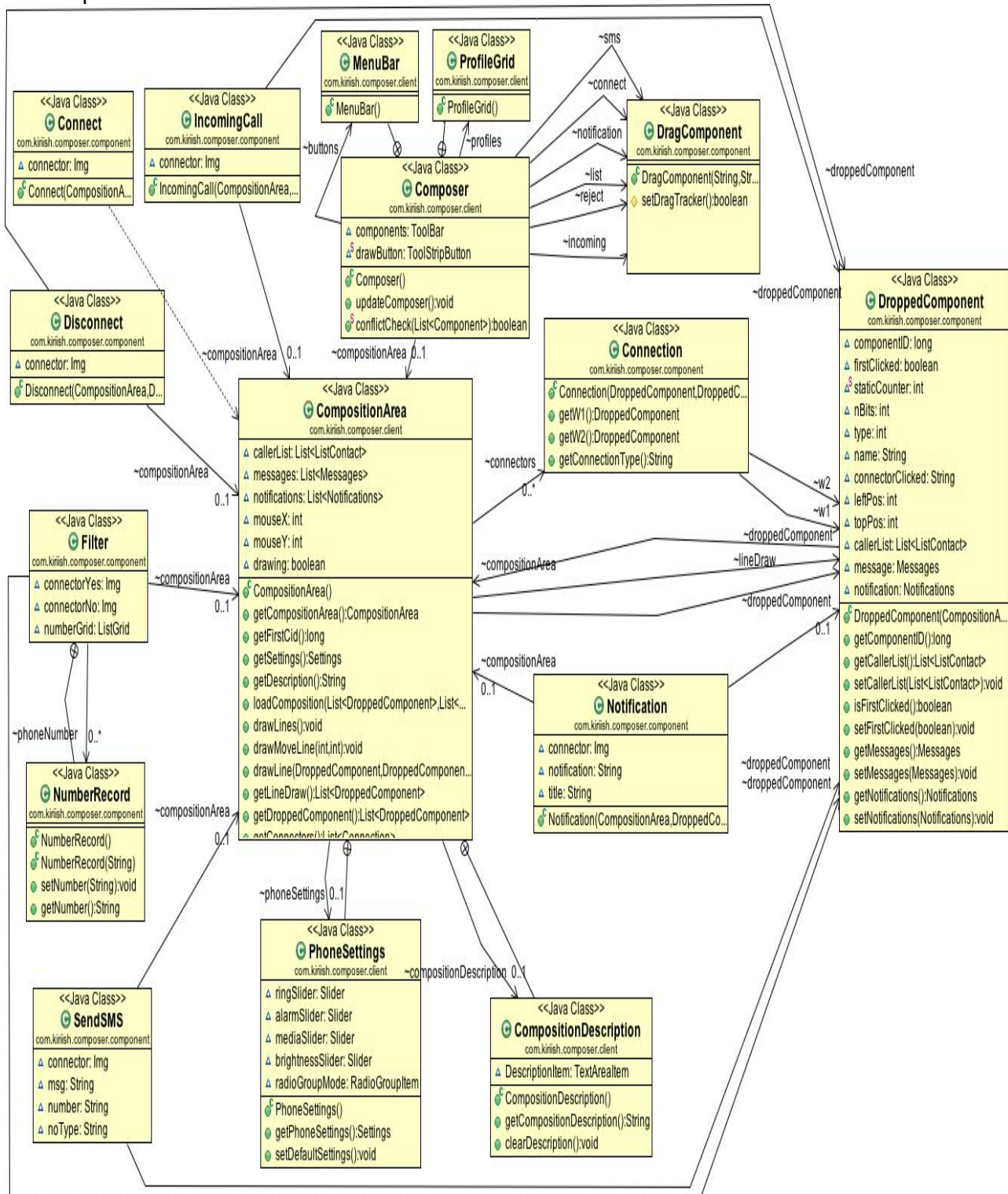


Figure 4-17: Class Diagram, Profiles functionality

The Profiles functionality resides in the above classes. The *Composer* class main functionality is to load the GUI with the classes *CompositionArea*, *PhoneSettings*, *CompositionDescription*, *ProfileGrid* and *MenuBar*. A description of the class diagram is presented below:

- The *MenuBar* has the functionality of creating, saving and deleting a composition. These changes are stored in the *ProfileData* class as well as the database.
- The *ProfileGrid* class takes care of the representation of the already created profiles. When a record is clicked the selected composition is loaded from the *ProfileData*, and displayed in the composition area with the phone settings and description. As we can see from the Profiles GUI in Figure 4-3, there are 6 components that can be dragged from a bar on to the composition area. These components are instances of the *DragComponent* class.
- The *CompositionArea* class is where the composition area GUI is loaded. Here the components are dragged, dropped and connected to each other by the user. The composition area creates instances of the classes *DroppedComponent*, *Connection*, *PhoneSettings* and *CompositionDescription* when a new composition is created.
- Instances of the *DroppedComponent* class are created when a *DragComponent* object is dragged and dropped in the composition area. When a component is dropped a method checks what type of component it is and creates an instance of the component dependent on the type. The component is then added to the composition area. The arrows that connect components are called connectors and are instances of the class *Connection*.
- The instances of the *DroppedComponent* and *Connection* classes are stored in a list. The position where the *DroppedComponent* is moved is saved as well as the type, and which connectors that are connected to it. The *Connection* class creates to instances of the *DroppedComponent* class as well as the connection type and the graphical presentation of the arrow between the components. The *DroppedComponent* instances can naturally only be the same types as the six component types in the *DragComponent* class.
- As we have seen the *CompositionArea* class stores information about components and how they are connected. Furthermore, the *CompositionArea* class also needs to store information on each component, settings and description. Each component has its own class: *IncomingCall*, *Connect*, *Disconnect*, *Filter*, *SendSMS* and *Notification*. These classes have the same functionality as the related components in the previous system. The *Filter* class contains information about the *Caller List* component. All of the classes contain information about the graphical presentation and the functionality when the component is clicked.
- The *Filter*, *SendSMS* and *Notification* classes also contain extra information. The *Filter* class contains information of the list of number records that are instances of the *NumberRecords* class. The *SendSMS* and *Notification* classes contain information of the message, phone number and title. Furthermore, the phone settings and description of the profile are instances of the *PhoneSettings* and *CompositionDescription* classes. The *PhoneSettings* class contains information about the ring, alarm, media and brightness slider and also

which mode (normal, vibrate, silent) the phone should be in. The *CompositionDescription* class simply contains a text field where the user can type in a description. In addition, the *CompositionArea* class have methods for deleting the components and the graphical representation of drawing the arrows etc.

- The *Composer* class has a method called *conflictCheck()* , which is activated when a composition is saved. The *conflictCheck()* takes a list of *Components* in the composition and checks if there are any unconnected components, wrong connected components or multiple instances of certain components. The conflict resolution will be described later in this chapter.
- The *CompactComposer* class is almost identical to the *Composer* class. The small screen Composer for smart phone screen sizes called *ComposerAndroid* works a bit different. In this class, an array is created for the tree-structured composition. The data for the Notification, SendSMS, Caller List and phone settings are saved outside this array. If the composition is saved a method called *convertToXML()* is called to create an XML from the array and the data saved outside the array. This XML is identical to the XML files created in the *Composer* and *CompactComposer* classes.

In this section the Profiles functionality has been described. In the next section the Map functionality is presented.

4.2.2.3 The Map Functionality

In this section the Map functionality is described. A description of the classes related to the Map functionality showed in the class diagram in Figure 4-18 follows below:

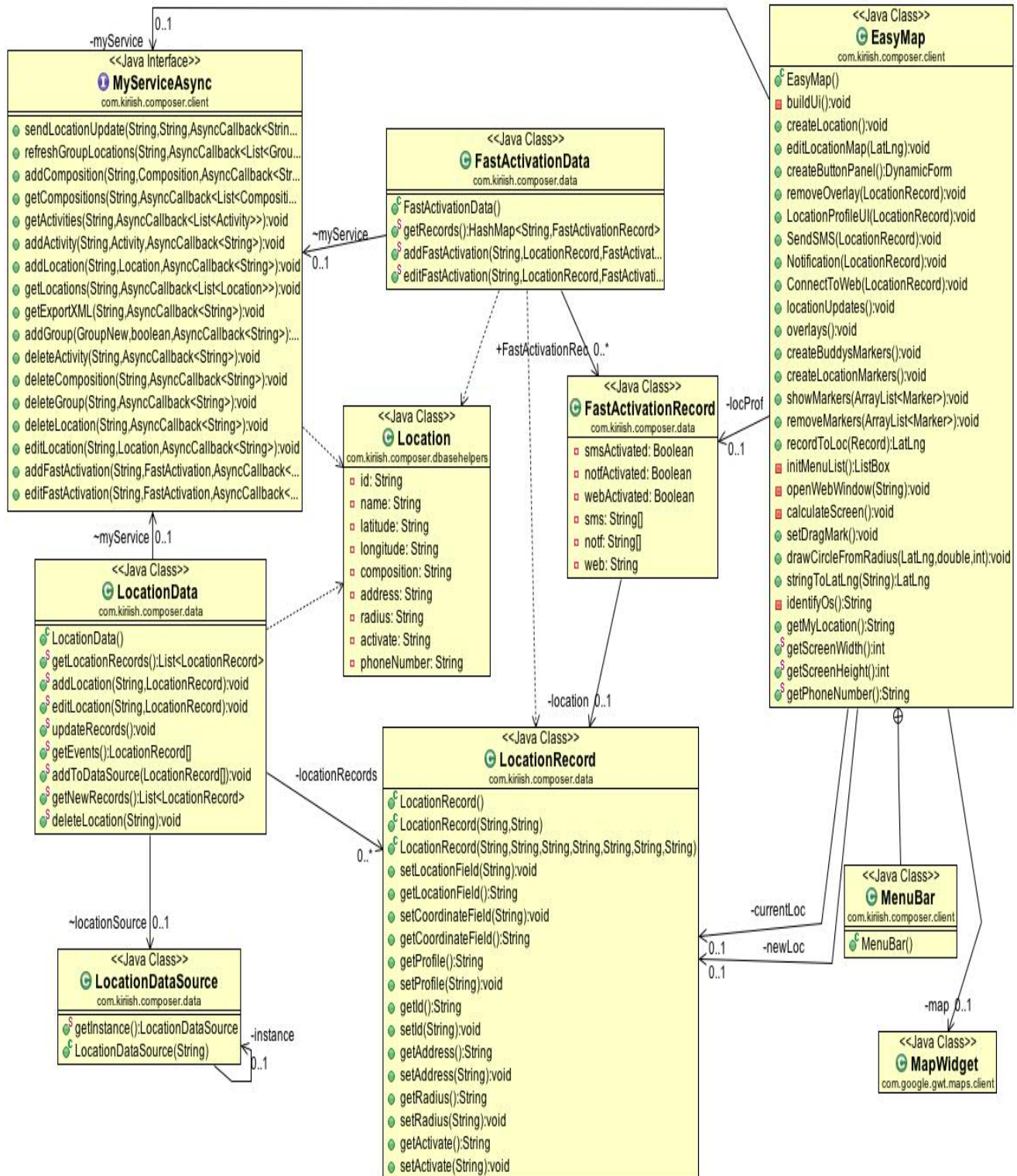


Figure 4-18: Class Diagram, Map functionality

A description of the class diagram is presented below:

- The Map functionality resides in the *EasyMap* class. Initially the *EasyMap* is loaded with a *MenuBar*, map area and a bottom menu with buttons as seen in Figure 4-6.
- The map area contains a *MapWidget* from the GWT library and the size of the *MapWidget* is set relative to the screen size. The screen size is obtained by calling the JSNI methods *getScreenHeight()* and *getScreenWidth()*.
- The location drop down grid, where the locations can be selected is initially populated from the *LocationDataSource*, which is set in the *LocationData* class method *addToDataSource()*. The *LocationData* class contains a list of *LocationRecord* objects. As previously mentioned, the *LocationData* class populates this list during the start up functionality in the *EasyDroid* class. When a location is selected in the location drop down grid, the *MapWidget* sets the centre of the map to the coordinates of the location selected.
- When the user wants to add a location he/she can either search for addresses or manually set the location on the map with the *dragable* marker feature. The search feature utilizes the *GeoCoder* widget from the GWT library, which takes a string as input parameter and returns the address coordinates. If the search is successful, the *MapWidget* sets the map centre to the address coordinates and the *dragable* marker feature is activated by calling *setDragMark()* method.
- When the user is saving a location, the method *createLocation()* is called. The *createLocation()* calls the method *addLocation()* in the *LocationData* class, which takes care of storing the location in the database. If it stored successfully, a new marker is created with the coordinates of the *dragable* marker, or at the centre of the map if the *dragable* marker feature is not activated. The storing is done in the same manner as in the Profile functionality.
- Locations that are added can be edited or deleted in the location settings window by clicking a marker as showed in Figure 4-8. The location settings and the previously mentioned Fast Activation functionality with the SendSMS, Notification and Web Connect features can be set in this window.
- This information is saved in a *FastActivationRecord* object. If the user chooses to save it, the *LocationData* class method *editLocation()* is called with the *LocationRecord* object as a parameter. Inside the *editLocation()* method a *Locations* object is instantiated and set with the appropriate fields from the *LocationRecord*. In addition, the methods *editFastActivation()* or *addFastActivation()* is called in the *FastActivation* class to save the FastActivation settings for a location.
- Furthermore, these methods call the *MyServiceAsync* method *editLocation()* and *editFastActivation/addFastActivation* with the *Locations* and *FastActivation* objects. The database will now be updated with the latest edition of the location. The *LocationData* class also takes care of deletion of a location when the remove button is clicked. The corre-

spondent Fast Activation settings for the location are also deleted. This will be described when the server side of the system is presented. Notice that the GUI is not updated if the changes are not saved in the database. Furthermore, a *LocationRecord* cannot be passed directly since the existing database is not set up to handle this object yet.

- The Map area has the option of showing both location markers and the Group members locations as seen in Figure 4-9. This is done by calling the method *Overlays*, which calls the methods *createLocationMarkers()* and/or *createBuddyMarkers()* dependent on which boxes the user has checked.
- The *createLocationMarkers()* gets its locations from the list of *LocationRecords* in *LocationData* class, and create markers from these locations.
- The *createBuddyMarkers()* method works similar, but receives its group member's location from the list in the *GroupData* class. When the markers are created the *showMarkers()* is called to put the markers on the *MapWidget*. If the user unchecks the check boxes, the *removeMarkers()* is called to remove the markers from the *MapWidget*.
- The user can show his location in the Map by calling the JSNI method *getMyLocation()*, if the application is loaded on an Android device. This method will then be called on the Android Engine and the location will be sent back to the GWT Interface.
- The users location can then be shared to the server by calling the *locationUpdates()* method as seen in Figure 4-10. The location is sent to the server by calling the *sendLocationUpdate()* in the *MyServiceAsync* class, which takes the user phone number and coordinates as a parameter. If the box for periodically sharing the location is checked a timer is initiated, which calls the *sendLocationUpdate()* every 5 seconds.

In this section the Map functionality has been described. In the next section the Calendar functionality is presented.

4.2.2.4 The Calendar Functionality

This section presents the Calendar functionality. The classes related to the Calendar functionality are showed in the class diagram in Figure 4-19 and is followed by a description:

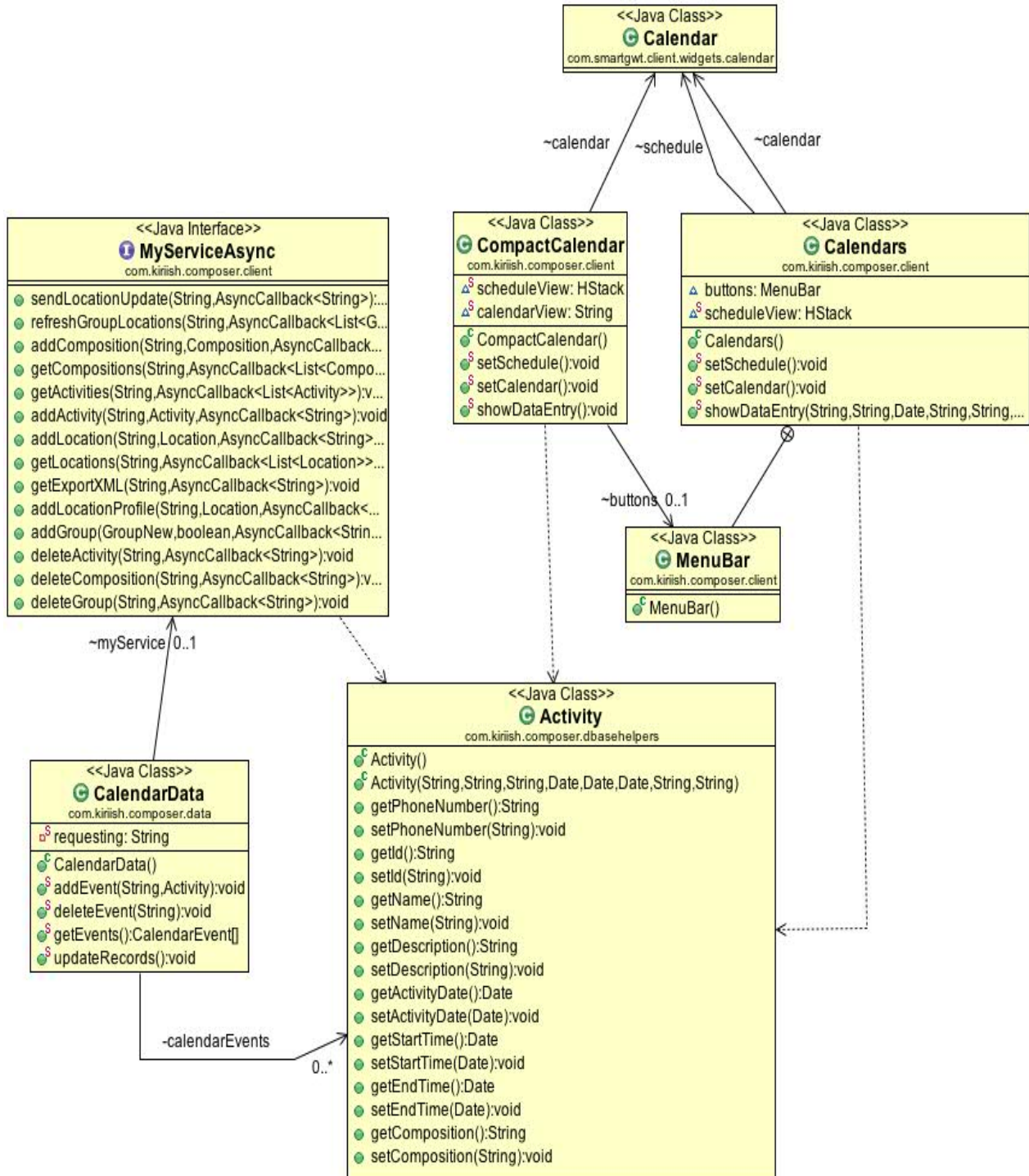


Figure 4-19: Class Diagram, Calendar Functionality

A description of the class diagram is presented below:

- The *Calendar* class make use of the Calendar widget from the SmartGWT library. The class creates two object instances of the Calendar widget.
- The first object is used as a mini calendar as seen to the left in Figure 4-11, by calling the method *setCalendar()* on the object. The *setCalendar()* method sets the appropriate attributes of the widget. The second object is used a time schedule, by calling the *setSchedule()* method on the second object. The *setSchedule()* method works in the same way as the *setCalendar()* but sets different attributes. The result is the schedule seen to the right in Figure 4-11.
- The *CompactCalendar* works almost in the same way as the Calendars. However, the *CompactCalendar* only creates one instance of the Calendar widget. As we can see from Figure 4-12, the *CompactCalendar* class initially loads a large version of the mini calendar, by calling the *setCalendar()* on the object. When a date is clicked, the *setSchedule()* method is called on the very same object. The widget will then be transformed into a schedule as showed in Figure 4-12.
- Both of the *Calendar* and *CompactCalendar* classes contains an instance of the *MenuBar* class. The *MenuBar* class has one button for creating a new event in the calendar. When this button is clicked, the method *showDataEntry()* is called.
- The *showDataEntry()* method creates a new window as showed in Figure 4-11, where the user can set the calendar event information. If the user chooses to save the event, a new instance of the *Activity* class is created, where the activity name, description, date, time and profile are set, as well as the users phone number.
- In order to save the event in the database the *CalendarData* class method *addEvent()* is called with the *Activity* object as a parameter. This method will then call the method *addActivity()* in the *MyServiceAsync* class, which makes an asynchronous call to the server with the *Activity* object as a parameter and a string as the callback. If the storing in the database is executed successfully, the callback returns the string “True” and the event are saved in the *Calendar* schedule. By doing the saving in this manner, the database and the GUI will always be synchronized. The saving is carried out in the same manner as the Map and Profiles functionality.
- The *CalendarData* class contains a list where the *Activity* objects are stored. As previously mentioned, the *EasyDroid* class calls the *CalendarData* class method *updateRecords()* in order to populate this list by calling the *getActivities()* method from the *MyServiceAsync* class.
- The *CalendarData* class also takes care of the editing and deleting of activities from the activity list, as well as the database. The communication and sequence of calls will be described after the server part is presented.

In this section the Calendar functionality has been described. The next section presents the Groups functionality.

4.2.2.5 The Groups Functionality

In this section the Groups functionality is described. The class diagram below shows the related classes to the Group functionality:

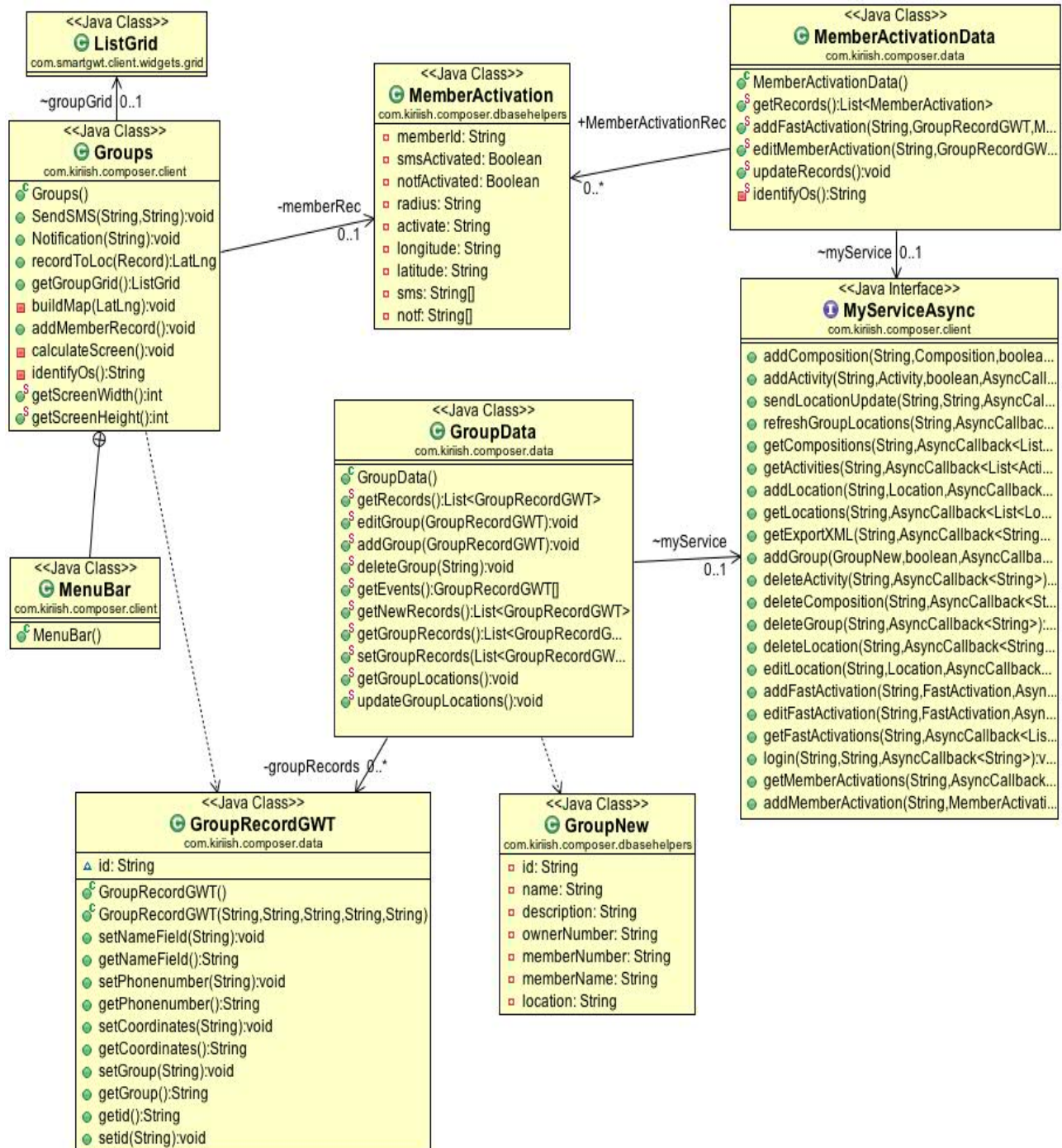


Figure 4-20: Class Diagram, Groups Functionality

A description of the class diagram is presented below:

- The Groups functionality consists of the *MenuBar*, group grid and two buttons beneath the group grid as seen in Figure 4-13.
- The group grid is a *ListGrid* widget from the GWT library. Initially, the Groups class is initiated and the group grid is populated from a list of *GroupRecordGWT* objects.
- The user can add a new member to the grid by clicking the *Add Member* button, and a window similar to the one in Figure 4-13 appears. When the *save* button is clicked, the *addGroup()* method in the *GroupData* class is called, which consequently calls the *addGroup()* method in the *MyServiceAsync* class in order to store the record in the database. If the record is stored successfully the group record is added to the list of *GroupRecordGWT* objects, and is showed in the group grid. The saving is done in the same way as the other main functionalities.
- The user has the option to edit the fields of the group record in the group grid. If the field change from the old value, the *GroupData* method *editGroup()* is called, which in turn calls the *addGroup()*. Editing has the same set of action sequences as for adding, except that the editable field set to “true”.
- If the user wants to delete a record the user can press the settings icon and the remove button as showed in Figure 4-14. The call sequence is the same as for adding a member, except that the *deleteGroups()* methods in *GroupData* and *MyServiceAsync* classes are called.
- The group members also have the Member Activation functionality attached to the group members location. The Member Activation functionality can be seen in Figure 4-14, and works almost in the same way as the Fast Activation functionality. If the save button is pressed the methods *addMemberActivation()* or *editMemberActivation()* is called in the *MemberActivation* class. These methods will call the correspondent methods in the *MyServiceAsync* class and update the database.
- The user can also display the group member location by clicking the “Member Location” button. When this button is clicked, a *MapWidget* showing the member location with a marker is displayed. The user can obtain the group members latest location by clicking the “Refresh” button. The method *UpdateGroupLocations()* will then be called in the *GroupData* class. This method receives the updated group member locations by calling the *refreshGroupLocations()* in the *MyServiceAsync* class. This will be described in greater detail when the Buddy List functionality is presented.

In this section the architecture and implementation of the GWT GUI was presented. The application architecture can be divided into the main classes with the GUI functionality, classes for handling the saving and retrieval of data, the Server and Android communication classes, and some helper classes. This is a good structure of the application architecture. The implementation still has some potential to be more optimized and structured. However, the GWT compiler does code optimisation when compiling to JavaScript. The next section presents the EasyDroid Engine.

4.2.3 The EasyDroid Engine

The EasyDroid Engine is an Android application written in native Android code. It is here the activation of profiles and the Fast and Member Activation features are actually triggered. The Engine receives information from the server database, which it populates its own database with. The EasyDroid Engine is a reimplementaion of the existing EasyDroid Engine without the GUI elements and other unnecessary parts. Since the Engine and GUI was strongly dependent, removing only the GUI part seemed cumbersome and time consuming. Hence, the EasyDroid Engine was made from scratch, but uses many of the existing EasyDroid classes. The description details are kept to a certain level in order to not confuse the reader, but also give a good understanding of what is going on. Some methods are not described, as they are not used anymore or not important. An overview class diagram of the EasyDroid Engine is presented here:

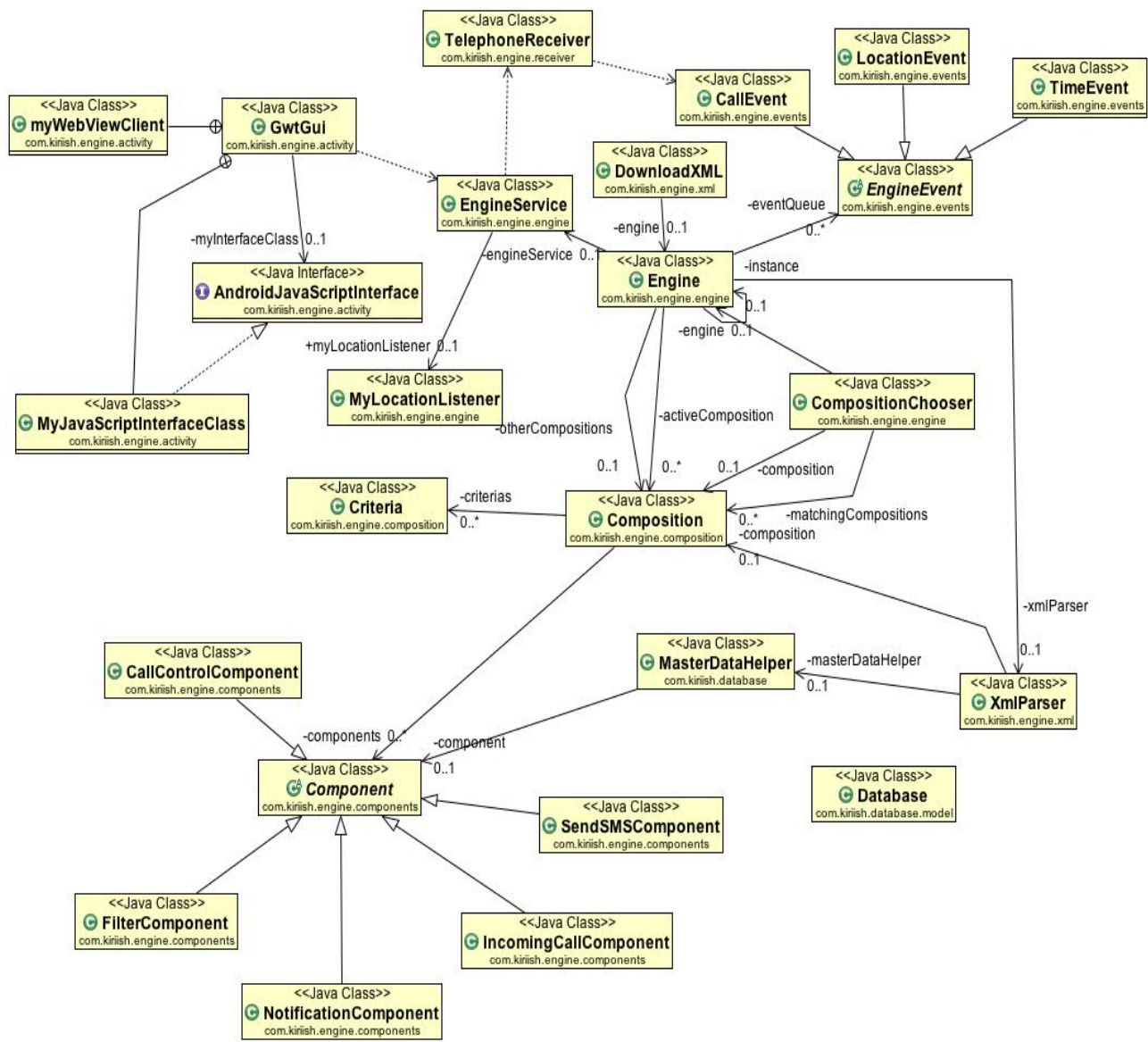


Figure 4-21: Class Diagram, EasyDroid Overview

In Figure 4-21, the main classes of the EasyDroid Engine are presented without attributes and operations. We will take a deeper look into the different parts of the EasyDroid Engine. It is natural to start with describing the applications start up classes.

4.2.3.1 The Start-Up Functionality

A description of the class diagram in Figure 4-21 is presented below

- The *GWTGui* class is the initial class of the EasyDroid Engine, which loads the GWT GUI by utilizing the *WebView* class.
- The *WebView* object function is to load web sites by its URL[22]. *WebView* is a part of the Webkit library, which is used for mobile web rendering on Android devices etc. By using *WebView* the application can view the same HTML content as the browsers on smart phones [23].
- The *GWTGui* class also implements the *AndroidJavaScriptInterface* in a class named *MyJavaScriptInterfaceClass*. Furthermore, *MyJavaScriptInterfaceClass* functions as the communication link between the GWT GUI and the EasyDroid Engine. It contains different methods that can be called from the GWT GUI. The methods are the following: *getMyLocation()*, *setXML()*, *setFastActivation()* and *setMemberActivation()*.
- The *getMyLocation()* method gets the device location from and sends the location coordinates to the GUI.
- The *setXML()* gets the *Composition XML* file from the GWT GUI and populates the database by calling the *parseXML()* and *loadFromDatabase()*. An example of the *Composition XML* is showed in Appendix C.1. This will be described in detail later.
- The *setFastActivation()* and *setMemberActivation()* gets the Fast and Member Activation XML files from the GWT GUI and populates the database and a list of Fast and Member Activation objects in the *Engine* class.
- The *WebView* object must add the *MyJavaScriptInterfaceClass* for the communication to work. The *AndroidJavaScriptInterface* communication will be described later in this chapter.
- Finally, the *GWTGui* starts the Engine Service that initiates the actual functionality of the Engine. Before we go on the engine we take a look at the composition with components in the section.

In this section the Start-Up functionality has been described. The next section presents the Composition functionality.

4.2.3.2 The Composition Functionality

In this section the Composition functionality in the EasyDroid Engine is described. Figure 4-22 shows the correspondent class diagram, followed by a description:

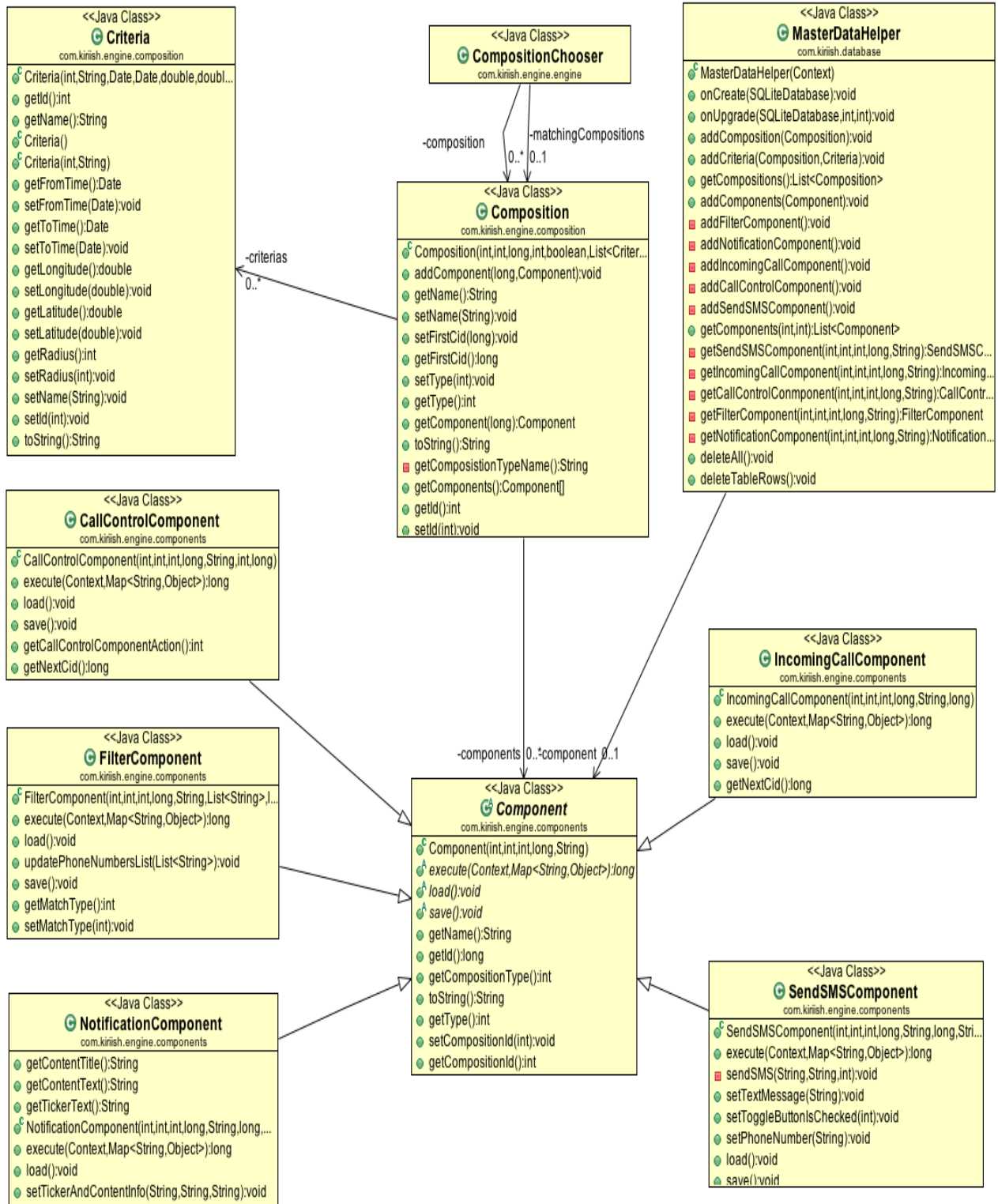


Figure 4-22: Class Diagram, EasyDroid Composition

A description of the class diagram is presented below:

- The *Composition* class contains a list of *Component* objects and a list of *Criteria* objects. *Composition* instances can be either a call type or a location type. In addition, the *Composition* instance contains the first component id of the composition and a priority attribute.
- The first component id is for knowing which component is the first in the composition. The next components in the composition are obtained by checking the *nextId* attribute of the *Component* object. The priority parameter is for determining which composition should be active when more than one composition is activated at the same time.
- The *Criteria* class represents a set of criteria required for a given composition to be activated. This could either be the latitude, longitude and radius if the composition is activated on a location or the time period if the composition is triggered by a time.
- The *Component* class is an abstract class, which the *CallControlComponent*, *FilterComponent*, *NotificationComponent*, *SendSMSComponent* and *IncomingCallComponent* classes inherits from. Each of these component classes inherits the *execute()* method which does the actual execution on the device based on which component it is. The *IncomingCallComponent* only returns the id of the next component when the *execute()* method is called.
- Similarly, the *FilterComponent* only returns two next component id's. When calling the *execute()* method it returns the id of the next component based on the incoming number and the number list. If the incoming number match a number in the number list, the next component id is given for a match, and if the number does not match the next id is given for no match.
- The *CallControlComponent* takes care of the call handling based on the input parameters. This means that the *execute()* method does the connecting or disconnecting of calls.
- The *NotificationComponent* contains the ticker, title and message fields of a notification. The *execute()* method calls the Engine class method *showNotification()* in order to show the notification.
- The *SendSMSComponent* class contains the number and message of the SMS. The *execute()* method uses the *SmsManager* to send a SMS.

In this section the Composition functionality has been described. The next section presents the Engine functionality.

4.2.3.3 The Engine Functionality

In this section the Engine functionality in the EasyDroid Engine is described. Figure 4-23 shows the correspondent class diagram, followed by a description:

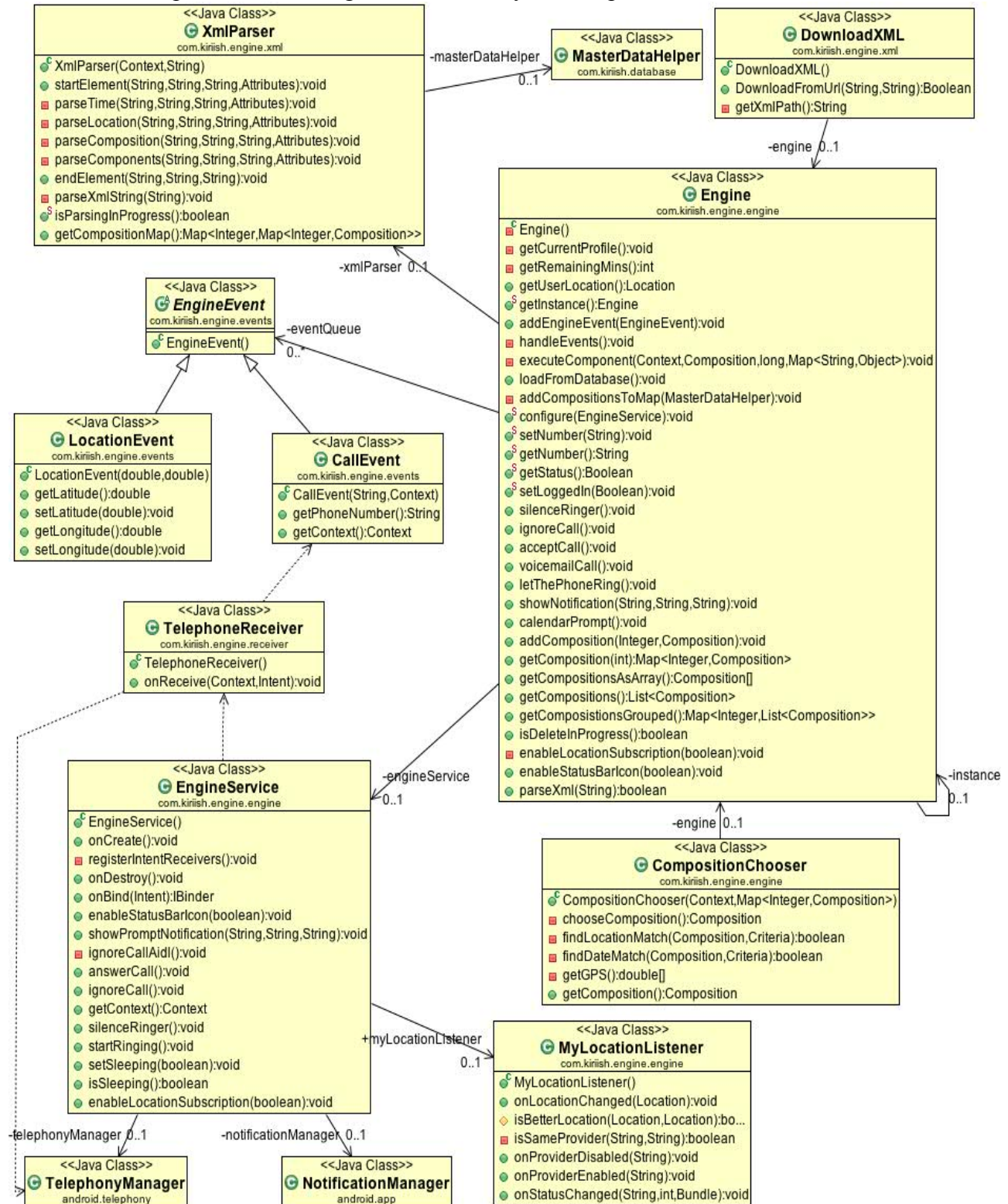


Figure 4-23: Class Diagram, EasyDroid Engine

A description of the class diagram is presented below. The *EngineService* and related classes is described first:

- The *EngineService* class is an Android Service, which means that it runs constantly in the background on the device.
- Initially, the *EngineService* calls the method *registerIntentReceivers()* for registering the *TelePhoneReceiever* to the application. The *TelePhoneReceiever* class extends the *BroadcastReceiever* class, which receives broadcasts from the application context. Basically, when the phone receives an incoming call, this receiver picks that intent up and adds a new event to the engine calling the *addEngineEvent()*.
- Furthermore, the *EngineService* creates a reference to the *Engine* by calling the *Configure()* method from the *Engine* with the *EngineService* instance as a parameter. The *EngineService* class initiates the *TelePhonyManager*, which provides access to information about the telephony services on the device.
- The *EngineService* can use the methods of the *TelePhonyManager* class to determine telephony services and do call handling [26]. The *EngineService* makes use of the methods *IgnoreCall()*, *IgnoreCallAid()*, *answerCall()* and *silenceRinger()* for the call handling.
- The *EngineService* class also initiates the *NotificationManager* class, which notifies the user of events that happens in the background. The *showPromptNotification()* method makes use of this class to show prompts of notifications that is triggered by events.
- Finally, the *EngineService* class subscribe to a location listener that listens for locations updates. The location listener is an instance of the class *MyLocationListener* class. *MyLocationListener* listens for location updates from GPS, Network and Passive location providers. Furthermore, it has a method *isBetterLocation()* for determining which location that is the most accurate of the location updates.

The *Engine* and related classes is described below:

- Initially, the *Engine* calls the method *parseXML()*, which takes the *Composition* XML downloaded from the GWT GUI as an input. The *parseXML()* method starts out with clearing the database by using the *MasterDataHelper* class. Then it initiates the *XmlParser* class to parse the XML file and puts the information into the database.
- Secondly, the *loadFromDatabase()* method is called. This method initiate the *MasterDataHelper* class and calls the *addCompositionToMap()* method with the instance as a parameter. The *addCompositionToMap()* retrieves the data from the database and puts it into two different Map structures of *Composition* objects.
- This results in a Map structure for profiles on locations called *allCompositionsMap*, and another for other profiles called *otherCompositions*. Now, the *Engine* class has the pro-

files stored in Map structures, which makes the look ups more efficient. The Engine also has a list of *EngineEvent* objects called *eventQueue*. These are the events that will trigger a profile to be active. The *addEngineEvent()* method is called for adding an event to the queue. As soon as an event is added, the *handleEvents()* method is called.

- The *handleEvents()* method iterates through the list *eventQueue*. For a single iteration, it first checks if the type of event is either a *CallEvent* or a *LocationEvent*. If it's a *CallEvent* it receives all the compositions from *allCompositionsMap* for the *CallEvent* type, if it's a *LocationEvent* it receives the compositions for *LocationEvent* types.
- The compositions is put it in a new map of composition objects. Then it creates an instance of the class *CompositionChooser*, which takes the Map of *Composition* objects as a parameter. The constructor of the *CompositionChooser* checks if there are any matching compositions to the current location or time, and returns the matching composition.
- In the *CompositionChooser* the methods *findLocationMatch()* and *findDateMatch()* does the checking. The *findLocationMatch()* method checks if the current location is within the radius of the composition criteria and the *findDateMatch()* checks if the current time is within the time criteria. If a matching composition is found the method *executeComponent()* is called.
- The *executeComponent()* method executes the components of the composition in a sequence based on the components *nextId* attribute as described in the *Composition* section.

The description above might seem a bit complex. Therefore, a sequence diagram for the method calls from triggering an event to the execution of components is presented below in a sequence diagram in order to create a better understanding for the reader:

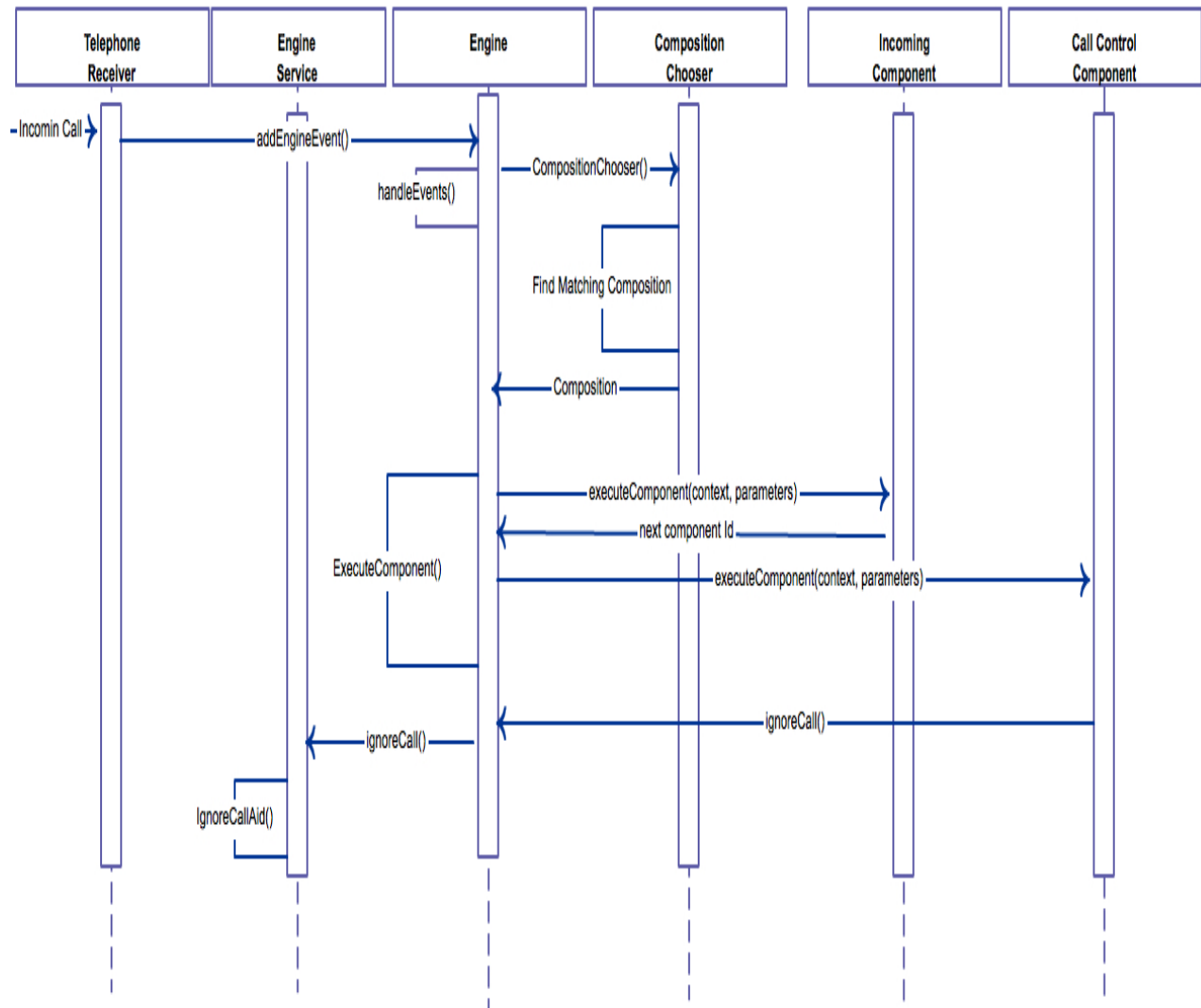


Figure 4-24: Sequence diagram, EasyDroid Engine

The figure above presents the sequence diagram for the activation of a simple composition disconnecting an incoming call.

1. Initially, the *TelephoneReceiver* class receives an intent telling there's an incoming call. The *TelephoneReceiver* class calls the Engine method *addEngineEvent()*.
2. The *addEngineEvent()* puts the event in the event queue and calls *handleEvents()* that will iterate over the event queue, handling each event.
3. The *handleEvent()* creates an instance of the *CompositionChooser* class and puts in the relevant compositions with the type *CallEvent*. The constructor of this class checks if the compositions criteria match any current location or time criteria for the device. In this case the *CompositionChooser* finds a match.

- The Engine calls the *executeComponent()* method on each component of the composition. As we can see from the figure the *IncomingComponent* class method *executeComponent()* is called and returns the id of the next component. The next component, the *CallControllComponent* class, is then executed. In this case the execution of this component calls the *ignoreCall()* in the Engine, which then calls the *ignoreCall()* method in the *EngineService* class.

The Fast Activation and Member Activation functionality are activated in the engine. Both of these functionalities are triggered by changes in the user location. A sequence diagram for the method calls with the Fast and Member Activation is presented below:

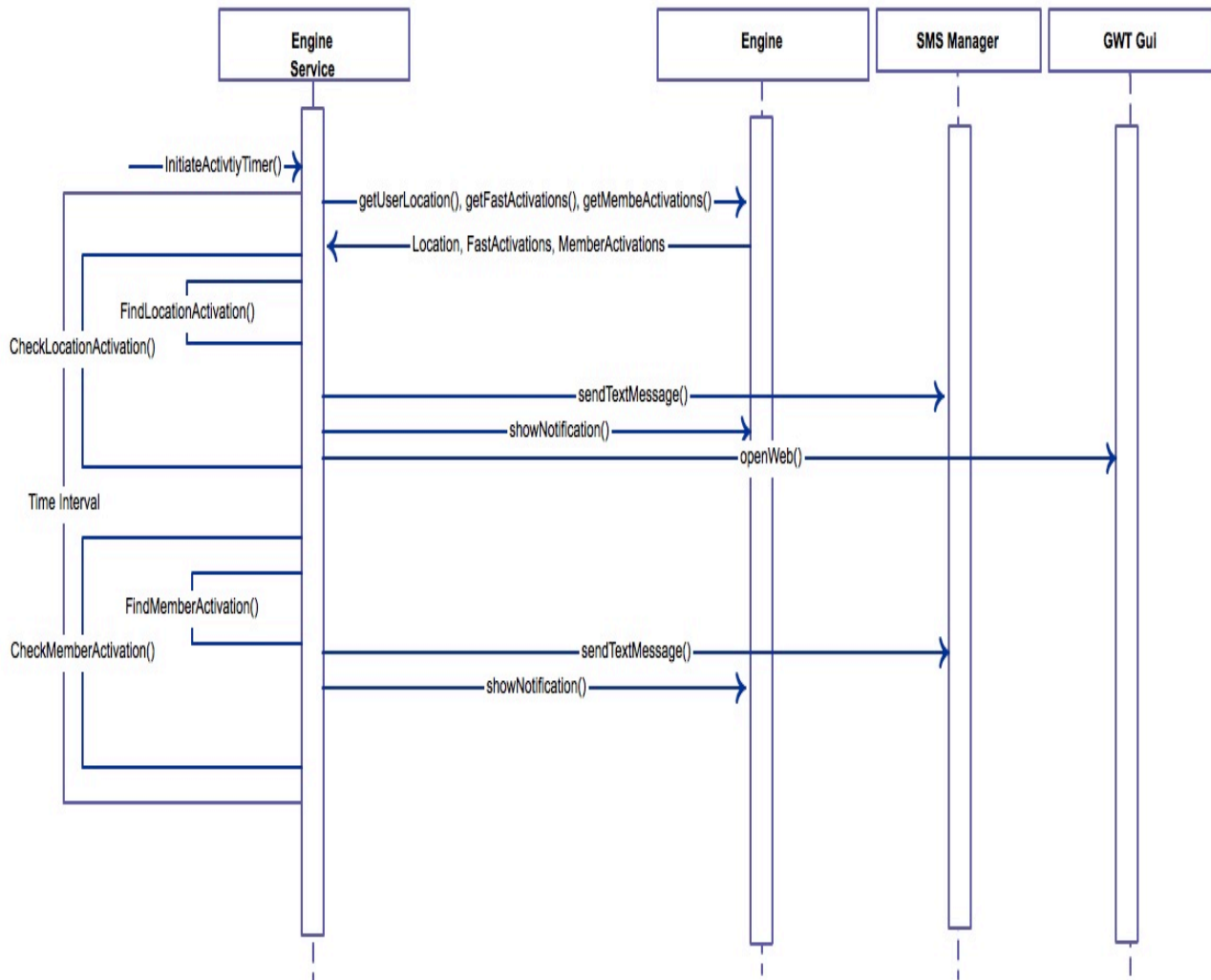


Figure 4-25: Sequence diagram, Fast & Member Activation

- Initially the *EngineService* class calls the *initiateActivityTimer()* method, which starts a timer that repeats every 15-second. In each time interval the *findLocationFastActivation()* and *findLocationMemberActivation()* are called.
- The *findLocationFastActivation()* checks if the users location is within the radius of the one of the Fast Activation objects location coordinates in the Fast Activation object list.

If a Fast Activation object is within the radius the method *checkLocationFastActivation()* method is called with the object as a parameter.

3. The *checkLocationFastActivation()* activates the Notification, SendSMS and WebConnect features depending of which feature that is activated. The Notification and SendSMS features are activated the same way as the *NotificationComponent* and *SendSMSComponent*. The SendSMS feature uses the *SMSManager* to send an SMS and the Notification feature calls the *Engine* class method *showNotification()*. The WebConnect feature is activated by opening a new *WebView* with the URL as parameter in the *GWTGui* class.

The Member Activation functionality works in the exact same way as the Fast Activation functionality. The only difference is that the member location coordinates will change since the members will move around as dynamic locations, as opposed to the static locations in the Fast Activation. Furthermore, the Member Activation functionality does not have the Web Connect feature.

In this section the EasyDroid Engine has been presented. The next section presents the server side of the system.

4.3 Server Side

The Server Side functionality has not been described thoroughly yet. Until now the system description has focused on the client part. In this section the server side of the new system is presented. The server side consists of the GWT server side and the database itself. The class diagram for the GWT server side is presented below:



Figure 4-26: Class Diagram, GWT Server side

The *MyServiceImpl* class is the implementation of the *MyService* interface on the client side of the GWT GUI. When a method is called on the client side, the implementation in the *MyServiceImpl* class executes through a remote procedure call (RPC). As we can see from the figure above, the *MyServiceImpl* class contains all the methods defined in the *MyService* interface as well as some other methods. Most of these methods use methods in the *DatabaseAccess* class for communicating with the database. The majority of the *MyServiceImpl* class methods is for adding, editing, deleting and getting calendar activities, map locations, composed profiles, group members, fast and member activation settings. These methods call the correspondent method in the *DatabaseAccess* class, which executes a query on the database. For instance, during the start-up, the application calls methods that will consequently call the *getLocations()*, *getActivities()*, *getCompositions()*, *getGroupMembers()*, *getMemberActivation()* and *getFastActivations()* for populating the GWT GUI with the data saved last time the application was used. Furthermore, the *addLocation()* method is called if a user adds a location to the map, and the *deleteActivity()* is called if an activity event is deleted from the calendar. In addition, the *MyServiceImpl* class also contains methods for logging in, getting and creating the XML that is parsed in the EasyDroid Engine and the authentication of group members.

In order to further describe the server side the database structure is presented below:

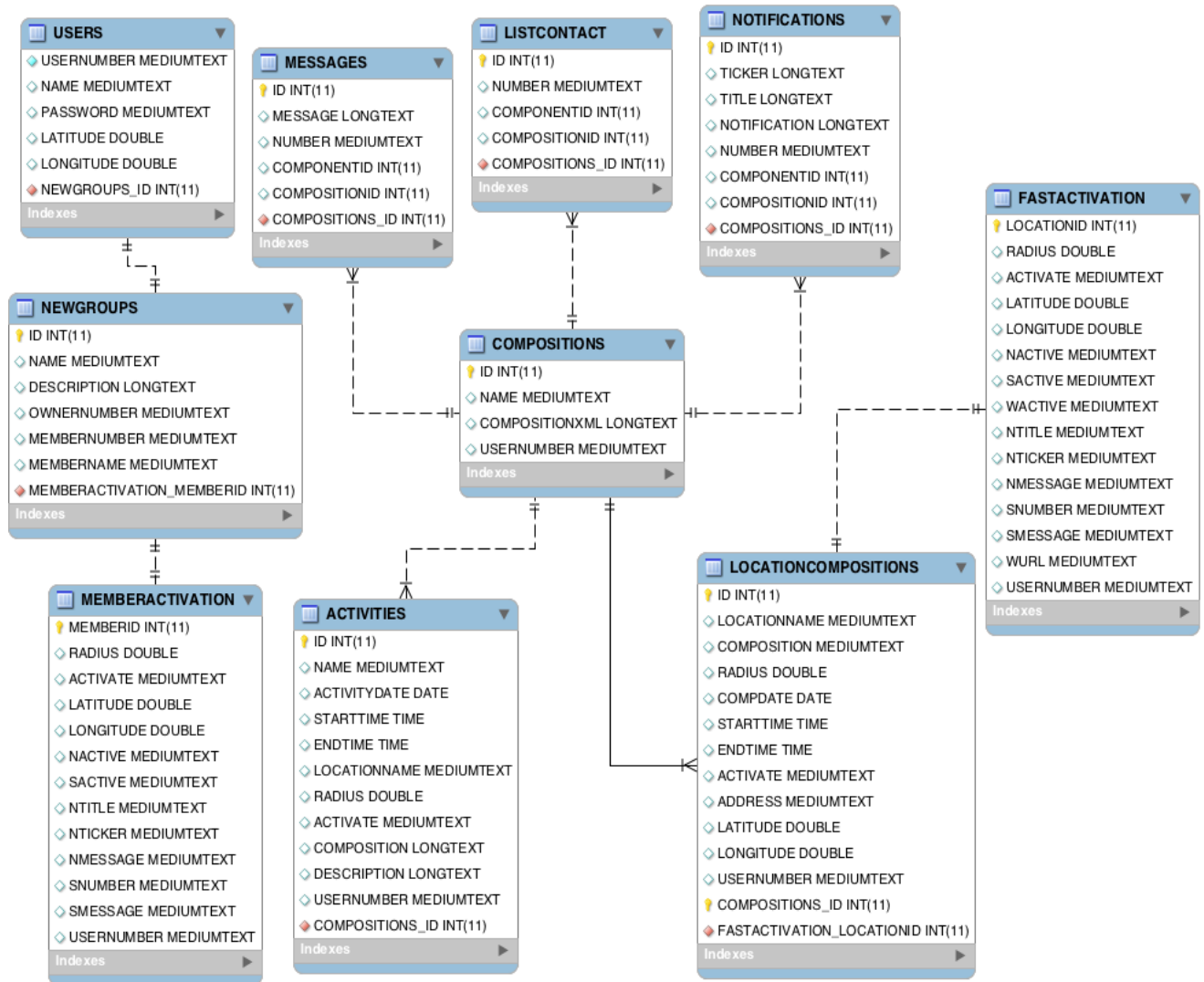


Figure 4-27: ER Diagram, Database

The database tables in Figure 4-27 ER diagram is described under:

- *Compositions*: The *Compositions* table is where the profiles area saved in the form of an XML string.
- *Messages*: The *Messages* table is where the *SMS* messages are stored. The *Messages* and *Compositions* table are connected by the *Composition_ID*. The profiles can have multiple *SMS* messages.
- *ListContact*: The *ListContact* table is where the caller lists are stored. The *ListContact* and *Compositions* table are connected by the *Composition_ID*. The profiles can have multiple caller lists.

- *Notifications*: The *Notifications* table is where the caller list is stored. The *ListContact* and *Compositions* table are connected by the *Composition_ID*. The profiles can have multiple caller lists.
- *Activities*: The *Activities* table is where the calendar events are stored. The *Activities* and *Compositions* table are connected by the *Composition_ID*. The profiles can be activated in multiple calendar activities.
- *LocationCompositions*: The *LocationCompositions* table is where the locations in the map are stored. The *LocationCompositions* and *Compositions* table are connected by the *Composition_ID*. The profiles can be activated in multiple locations.
- *FastActivation*: The *FastActivation* table is where the Fast Activation records are stored. The *FastActivation* and *LocationComposition* tables are connected by the *Location_ID*. A location can only have one Fast Activation record.
- *NewGroups*: The *NewGroup* table is where the group member records are stored. The *NewGroups* table is used together by the *Users* table to get retrieve a user location.
- *MemberActivation*: The *MemberActivation* table is where the Member Activation settings are stored. The *Member_ID* connects the *MemberActivation* and *NewGroup* tables. A group member record can only have one Member Activation setting.
- *Users*: The users table stores a users password and the users location. The Log In functionality makes use of the *Users* table.

The XML file that is eventually parsed by the EasyDroid Engine is created on the GWT server side. This is called the *Composition XML*. This is the file where the EasyDroid gets information on changes done to profiles, map locations and calendar activities in the database. When a change is done in the database, the resulting XML file will be different. Hence, the Engine needs to retrieve these changes each time the database is updated. The *MyServiceImpl* class method *getExportXML()* retrieves this XML by calling the *getExportInfo()* method in the *DatabaseAccess* class. The *getExportInfo()* method creates the XML file by doing queries to the *Activation*, *Composition*, *LocationComposition*, *Notification*, *Messages* and *ListContact* tables. The result from the queries gets XML tags appended, and an XML file is made. An example of the XML file is located in the appendix. As seen in Figure 4-26, the *MyServiceImpl* class has methods for adding, deleting, editing and retrieving the *FastActivation* and *MemberActivation* objects in the database.

The *DatabaseAccess* class also have two methods for creating the XML files containing the *FastActivation* and *MemberActivation* objects called *createFastActivationXML()* and *createMemberActivationXML()*. Examples of these files are presented in Appendix C.2 and C.3, These XML files that are downloaded and parsed in the EasyDroid Engine.

For the Log In functionality on the server side, the method *authenticateUser()* in the *DatabaseAccess* class is called from the *login()* method in *MyServiceImpl* class. The *authentica-*

teUser() queries the database for the password for the specific phone number and compares this with the input password. If the two password strings are equal, the user is authenticated.

Another form of authentication is done when a user wants to retrieve other users locations. A rule is made such that only users that have each other as members in their groups can retrieve each other locations. In other words, adding a user to one of your group's means that you give permissions to the added user for checking your location. This authentication is carried out in the *authenticateMembers()* method in the *MyServiceImpl* class. The *authenticateMembers()* method iterates through a list of members and is checking if each of the members has the requesting number in their list. If the members have the requesting number in their list, the member is authenticated and added to the authenticated member list. This method is called in the *refreshGroupLocations()* method to make sure the requested member locations are authenticated. The *refreshGroupLocations()* method is called each time the group's member grid in the GWT GUI is refreshed.

In this section the served side of the system has been presented. The next section describes how the different parts of the system interacts and communicate.

4.4 Usage, Communication & Interaction

In the previous sections the client side and server side of the new system is presented. This section describes the usage and communication of the entire system as well as some of the new functionality added to it. The first section describes the communication during normal usage of the system. The second and third section describes the new functionalities Fast Activation, Buddy List and Member Activation. The last section describes the Conflict resolution in the new system.

4.4.1 Communication & Usage

The system consists of the GWT GUI and EasyDroid Engine on the client side and the GWT Server Side and Database on the server side as seen in Figure 4-1. The GWT GUI communicates with the EasyDroid Engine through a common JavaScript Native Interface (JSNI). The GWT GUI communicates with the GWT Server Side with remote procedure calls (RPC). The GWT Server Side and the Database communicates through sockets with MySQL queries. The communication between the client and server side consists of adding, editing, deleting or retrieving data from the database. The majority of the communication between the GWT GUI and the EasyDroid Engine is sending updated XML files of changes in the server database and GUI. This communication is similar for all the Calendar, Map, Profiles and Groups functionality. The sequence diagram below describes this communication:

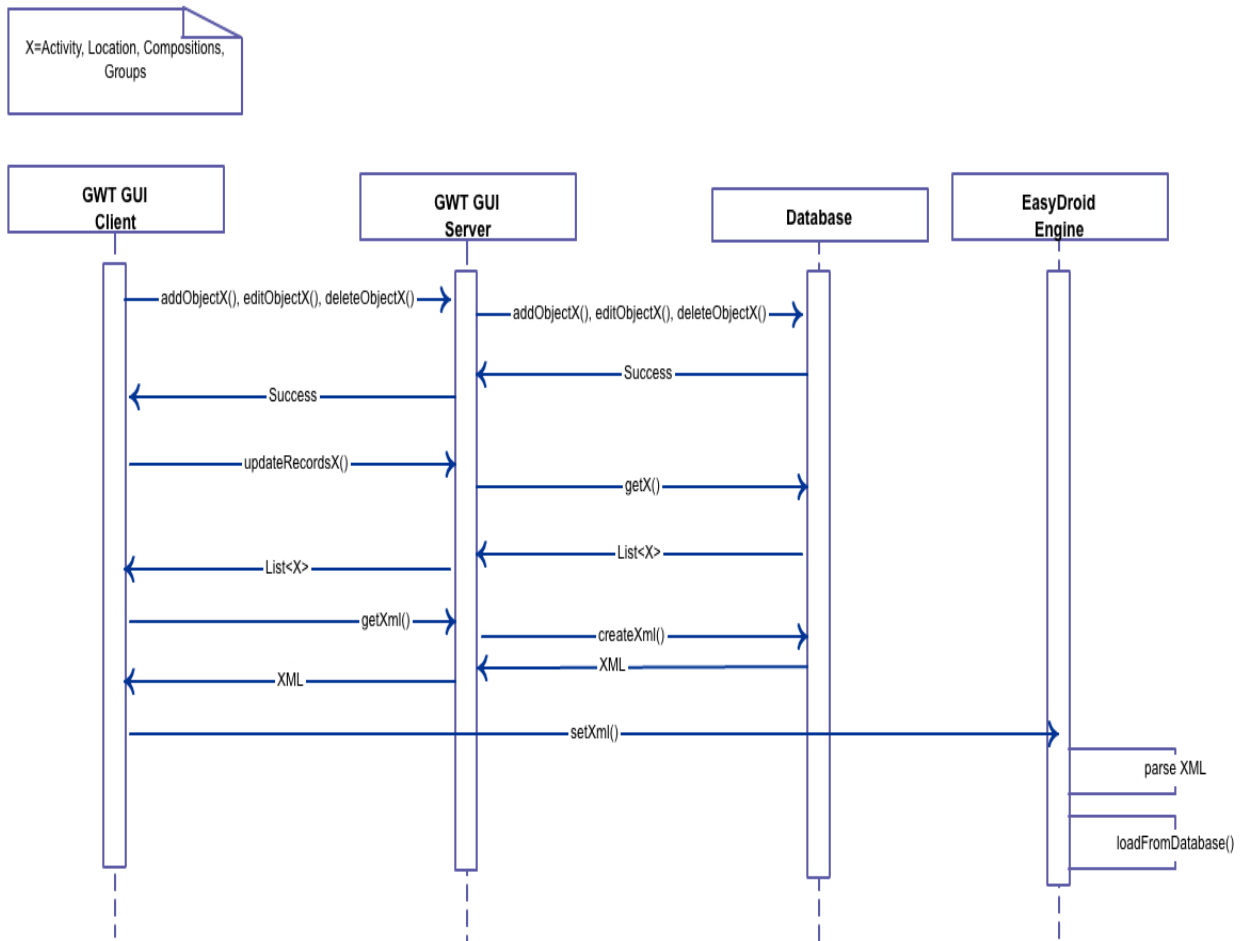


Figure 4-28: Sequence Diagram, System communication

The method calls in the sequence diagram in Figure 4-28 is common for the Calendar(Activity), Map (Location) and Profiles(Compositions) functionalities. The X in the methods can be an Activity, Location or Compositions object. For instance, the `addObjectX()` and `updateRecordsX()` methods do not exist but the `addActivity()` and `updateRecordActivity` does. A description of the sequence diagram is presented here:

1. Initially, the user can add, edit or delete an object X. The correspondent methods are then called on the GWT Server with a RPC in order to query the database on the server. If queries are executed successfully, the changes are saved in the GUI.
2. Secondly, the `updateRecordsX()` method is called to retrieve the list of recently updated objects X records in the database. Note that this method is also called initially in the application start up.
3. Furthermore, if the application is running on an Android device with the EasyDroid Engine installed, the Engine will fetch the updated XML file with new compositions activated on locations or activities. This XML is created in the `DatabaseAccess` class with

the method *createXML()*. If the application runs on a platform different than Android, the XML will be fetched the next time the Engine is started.

4. The *setXML()* method is called on the EasyDroid Engine with the XML as a parameter String. The Engine will know retrieve the latest changes in the GUI. This method is also called initially during the start up of the system.
5. Finally, the XML is parsed in the EasyDroid Engine. After the XML is parsed, it is saved to the database and then loaded from the database into the lists.

By communicating and saving in this manner, the GWT GUI, database and EasyDroid Engine are always synchronized.

The functionality described in this section can be used in a number of existing scenarios described in the previous Frank Mbaabu`s Master Thesis [21] and the UbiCompForAll Scenarios [24]. The next sections will describe new functionality added to the tool as well as some examples of usage.

4.4.2 Fast Activation

In the previous sections the Fast Activation functionality has been presented for the client and server side. This section will show how the Fast Activation is used all together with examples and a scenario. The next figure shows an example of the communication of the Fast Activation functionality:

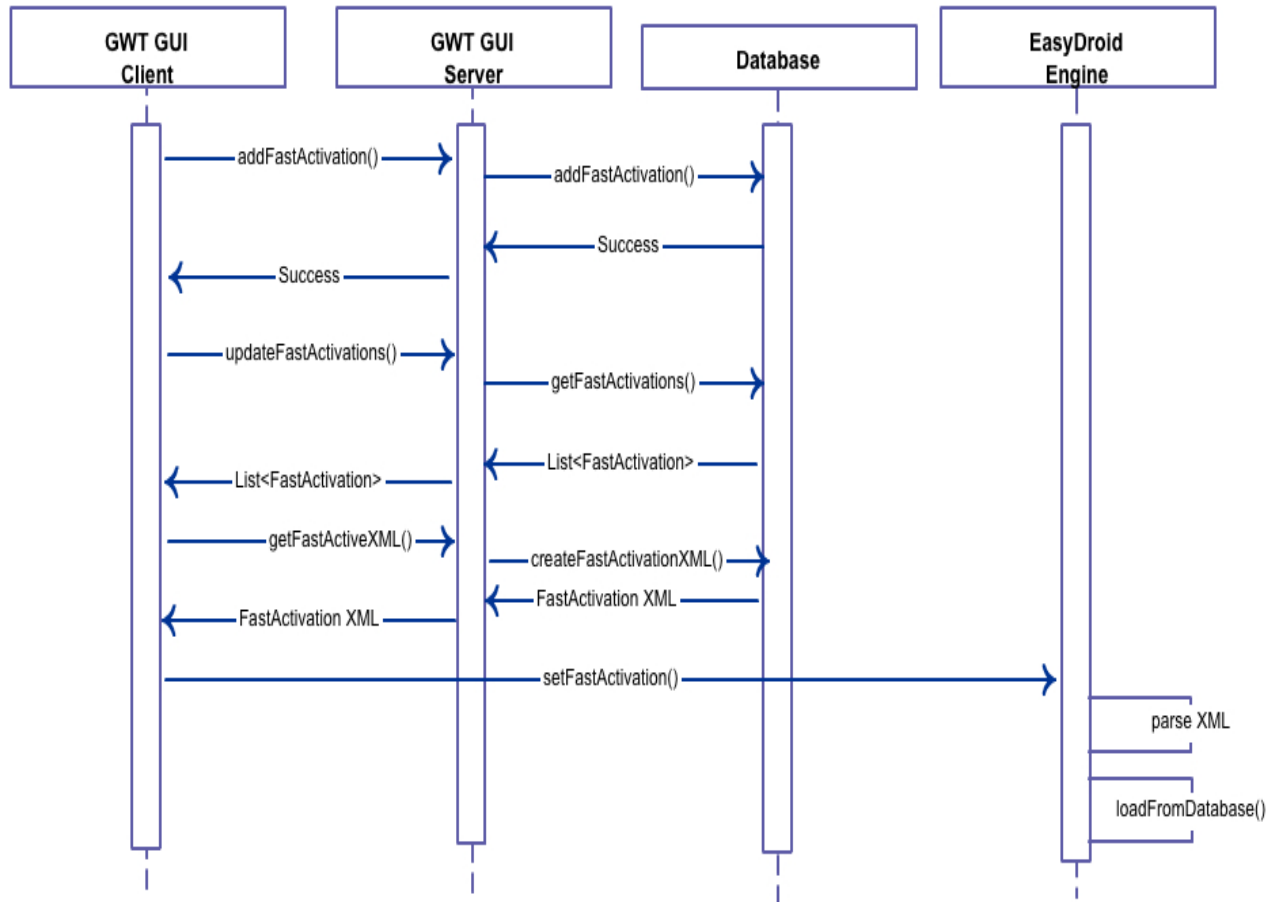


Figure 4-29: Fast Activation, Sequence Diagram

A description of the communication between the entities in Figure 4-29 follows below:

1. The Fast Activation settings are first uploaded to the database on the server. If the settings upload successfully, the settings are saved in the GUI.
2. Secondly, the `updateFastActivations()` method is called to retrieve the list of recently updated Fast Activation records.
3. Furthermore, if the application is running on an Android device with the EasyDroid Engine installed, the Engine will fetch the updated Fast Activation settings in a XML file. Otherwise, the Fast Activation settings will be fetched the next time the Engine is started.

4. The *setFastActivation()* method is called on the EasyDroid Engine with the XML as a parameter String.
5. Finally, the XML is parsed, saved to the database and then loaded from the database into the lists.

The Fast Activation functionality can be utilized in numerous of examples. The user can automatically send text messages when he/she is within the radius of a location. This could be practical if the user has a meeting loose time constraints. The other meeting participants will simply get a message when the user arrives. Another practical example is a package delivery scenario, where a coordinator wants to keep track of when the delivery persons has reached a delivery destination. Concerned parents would maybe like to now if their children has arrived at school or home. There are many examples of usage for the SMS feature alone.

The Notification feature of the Fast Activation functionality allows for automatic notification pop ups on locations. This can be helpful for users with a short-term memory or in unfamiliar environments. A user can be reminded with a notification if he/she is within the radius of a grocery store or a friend's house. If the user is located in an unfamiliar place and is easily disorientated, he/she can be notified when significant locations are nearby.

The Web Connect feature is a useful addition to the tool. The user can have the smart phone display any web site of his/her choice when he/she is within the radius of a location. This creates many new possibilities of location-based information displayed to the user. For instance, a user can obtain information about collective transportation for bus stops etc. Another example is displaying information about specified locations. This could be restaurants, concert clubs, historical venues etc. In fact, the user can create a custom-made city guide. This example is described further in the new city guide scenario presented next.

City Guide scenario: This scenario is inspired from the city guide scenario in the UbiCompForAll project [24]. In this scenario we follow Peter, a 28-year-old IT consultant from Oslo that is coming on a one-day visit to Trondheim. The problem is that he has never been to Trondheim before and wants information about buildings and sights when he walks around in the city. The time is of essence, and the goal is to get around quickly and obtain as much information of the city as possible. In addition he also wants to tell his friend when he is nearby his house to grab a quick lunch and get a notification of this. To achieve this goal he first looks up web sites that describe different building, sights, bus route schedules and also information about happenings at different places. Secondly, he logs into the GWT GUI and navigates to the Map functionality. Here he finds the location of each site he wants to visit in the map and adds the locations. For each location, he set the Fast Activation settings with the SendSMS, Notification and Web Connect features. For the different venues the Web Connect feature is activated and for the friends house the SendSMS and Notification features are activated.

When Peter is moving around in Trondheim, web sites will automatically pop up on the Android device if he is in the radius of one of the location. Web sites containing a huge variety of information are displayed. For example a bus route schedule when he is located at a bus stop, or the evening program/menu of particular clubs or cafes. He will also automatically send a text message to his friend when he is nearby and get a notification of this as well. The City Guide profile

will be of great benefit for Peter as he gets information about venues quickly and also get a sense of where he is located. Another benefit with the City Guide profile is that it is custom made by him. Using other city guides, much of the information could be uninteresting or irrelevant to his goals or objectives.

4.4.3 The Buddy List & Member Activation

The Buddy List concept was developed in the previous project thesis but was not integrated with the existing composer tool. The Buddy List System was a location-aware application developed for Android that could send and retrieve user location updates to a server. By obtaining information about other users context, it opens up more possibilities for where profiles can be activated and used. The most interesting information about the other users is perhaps their location. In the new system the Buddy List System is further developed and integrated into the tool. However, the implementation is a bit different as the client side functionality is spread over both the EasyDroid Engine and the GWT GUI. A simple overview of the new system is shown here:

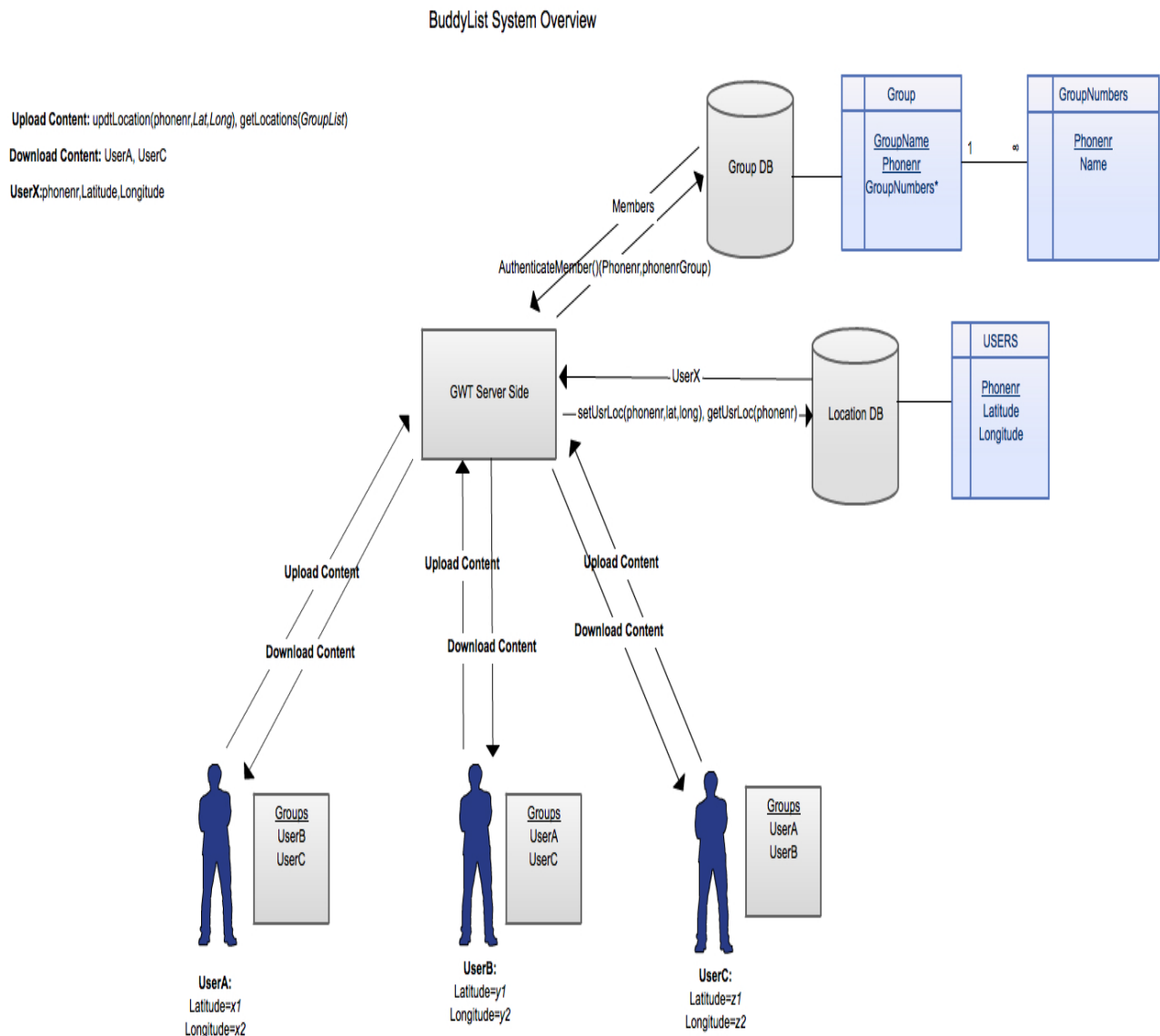


Figure 4-30: The Buddy List System

In Figure 4-30 the different users is the client side with the GWT GUI and the EasyDroid Engine. The Location Server and Database is on the server side. The device location is obtained

from the EasyDroid Engine in the MyLocationListener class. The MyLocationListener class listens for GPS and Network location updates and picks the most accurate location. This location is uploaded to the GWT GUI and can be showed in the Map GUI. The user has the option to either manually send this location to the server or set it to periodically send location updates with the Map functionality. The Groups functionality takes care of retrieving the other users location by obtaining the group members updated locations from the database. The next figure shows how the sending and retrieving of user location updates is done:

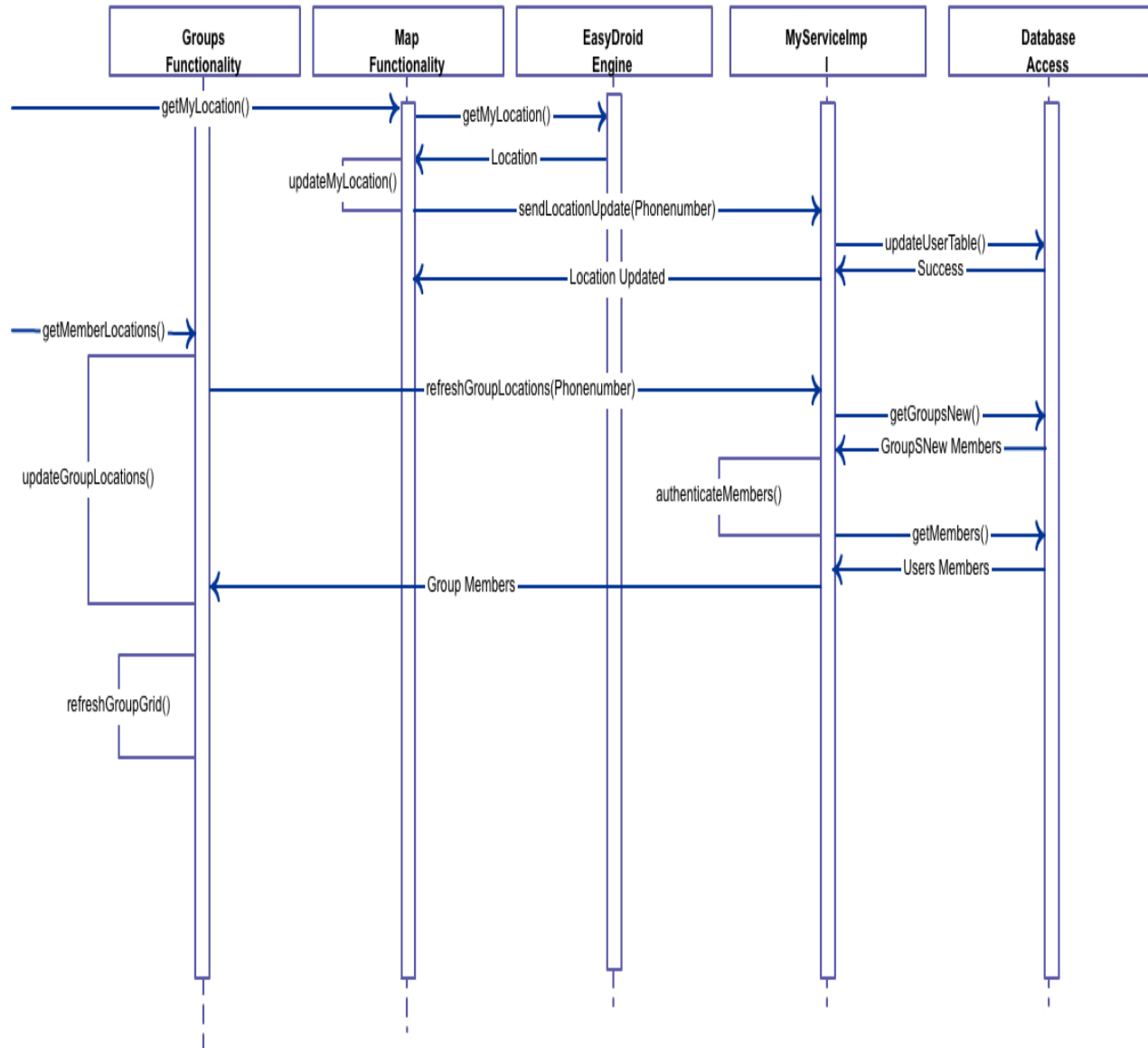


Figure 4-31: The Buddy List, Sequence Diagram

In Figure 4-31, the sequence diagram represents two scenarios of updating the users location and retrieving other users location. The updating a user location scenario is presented first:

1. Initially the *getMyLocation()* method is called in the Map GUI. This method will call the JSNI method *getMyLocation()* in the EasyDroid Engine.

2. In the EasyDroid Engine the most accurate location is obtained from the *MyLocationListener* class, and the location is returned as a string to the Map functionality.
3. When the location is obtained, the location is sent to the server either manually or automatically. Either way the method *updateMyLocation()* is called. This method calls the *sendLocationUpdate()* method in the *MyServiceImpl* class on the GWT Server side. The *sendLocationUpdate()* method calls the *updateUserTable()* method, which executes an update query on the database. Finally, the response is sent as a callback saying that the updating was successful.

The second scenario is for retrieving other users location:

1. Initially, the *getMemberLocations()* method is called in the Groups GUI in order to refresh the group member's location. This method calls the *updateGroupLocation()* method in the *GroupData* class.
2. Secondly, the *updateGroupLocation()* calls the method *refreshGroupLocation()* in the *MyServiceImpl* class in order to get the newest updated group members locations.
3. The *refreshGroupLocation()* calls the *getGroupNew()* method, which executes a query on the database, which retrieves all the group members for the specific phone number. Before the location can be retrieved the *refreshGroupLocations()* call the method *authenticateMembers()* to authenticate that the user can retrieve the members location. The member authentication was described in section 4.3. The *authenticateMembers()* method returns the authenticated members. Consequently, a new query is executed which retrieves the members with locations from the users table in the database.
4. Finally, the updated members are sent back and the group member's locations are refreshed in the Groups GUI.

The Buddy List also has a newly added functionality Member Activation. The Member Activation functionality is described in the previous sections and contains the SendSMS and Notification features. These features can be activated on a group member's location. The Member Activation functionality is showed in the sequence diagram below:

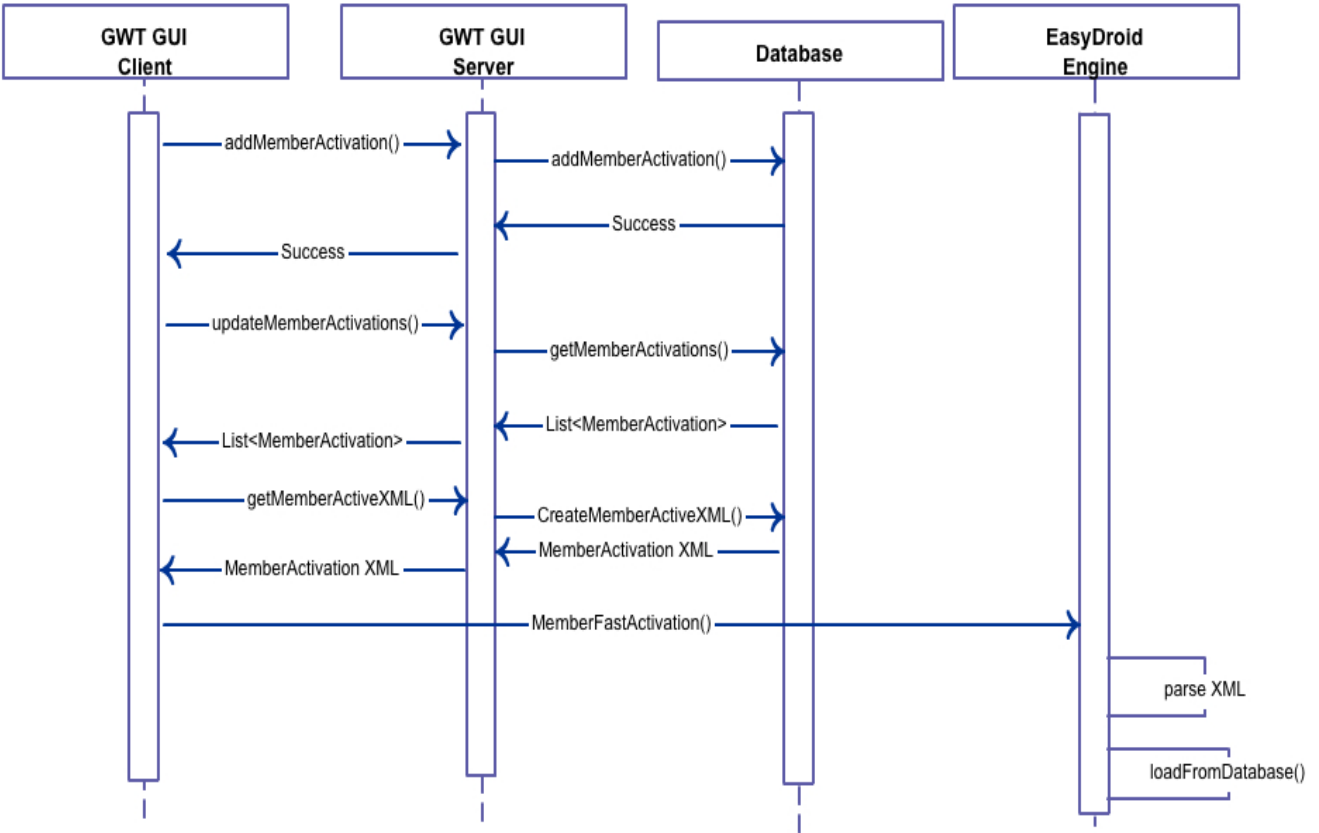


Figure 4-32: Member Activation, Sequence Diagram

The communication between the entities in Figure 4-32 is described here:

1. The Member Activation settings are first uploaded to the database on the server. If the settings upload successfully, the settings are saved in the GUI.
2. Secondly, the *updateMemberActivations()* method is called to retrieve the list of recently updated Member Activation records.
3. Furthermore, if the application is running on an Android device with the EasyDroid Engine installed, the Engine will fetch the updated Member Activation settings in a XML file. Otherwise, the Member Activation settings will be fetched the next time the Engine is started.
4. The *setMemberActivation()* method is called on the EasyDroid Engine with the XML as a parameter String.
5. Finally, the XML is parsed, saved to the database and then loaded from the database into the lists.

The Member Activation functionality improves the utility of the Buddy List system. The user can automatically send text messages when he/she is within the radius of a member location. This can be used in practical examples where members want to know when they are nearby each

other. This could be convenient for couples, friends or family members. Also varying professions where the workers are not stationary, could also utilize the SendSMS feature.

The Notification feature of the Member Activation functionality allows for automatic notification pop ups on member locations. In other words a user can be notified when a member is nearby. This could come in handy if the user wants to get notified when a group member is nearby, but not necessarily send them a SMS. A scenario is presented next to show an example of the Buddy List System usage.

Home Delivery Scenario: To show an example of how the Buddy List System can be utilized we use an example based on a “Home Delivery” scenario presented in [21]. In the home delivery scenario an online grocery delivery service is presented. The service is supposed to deliver groceries to people that has purchased the groceries online. In order to succeed, they need to plan well for their deliveries. The service therefore should keep track of where the deliverymen are located and also calculate the deliveryman closest to the delivery location.

The idea is that the person closest by handles deliveries within certain regions [21]. This refers to the second objective the scenario where the manager should be able to see the delivery persons’ locations in order to select the delivery persons closest to the delivery address to make the deliveries” [21]. The solution for this objective in the previous system is not sufficient. However the Buddy List system fits perfectly for this objective. Each delivery person can run the EasyDroid Engine that could periodically sends his or her location to the location server. The manager can then track their location in the Map GUI showing the group member as markers in the map. The manager can also set the Member Activity settings for each person such that he gets a notification when they reach a delivery destination. In this way the manager can keep statistics on each group member. Furthermore, the delivery persons can activate the FastActivation SendSMS functionality on each delivery person such that a text message is sent when the delivery person is nearby. The delivery persons could also activate the SendSMS or Notification functionality to inform the other delivery persons of their presence. With the new Member Activation and Fast Activation functionality added the tool the goals of the Home Delivery Scenario are more likely to be fulfilled.

4.4.4 The Conflict Resolution

In my project thesis a conflict resolution was proposed. This proposal has been developed further and implemented in the new system. An important feature of the new system is that the components are composed correctly in terms of usability, by preventing errors. When composing compositions in the Profiles functionality, many conflicts can occur. The composition can be wrongly composed, do not have any function, components that cannot be connected are connected etc. A new problem arises when a user has two different profiles that are activated simultaneously in the Map and Calendar functionality. This type of conflict occurs when a user is located in a location set with a profile, simultaneous as a profile set to a time and date is activated. In order to deal with these conflicts, there has to be a conflict resolution. In this section a solution for a more refined rule based conflict resolution for dealing with these types of conflicts is presented.

The previous version of the tool had some simple conflict checks for composing services in the Composition area. For instance, the Easy Composer did not allow for adding more than one component of certain components and some components could not be connected to each other. However, there were many conflicts that were undetected. Hence, a proposal for a conflict resolution was created in the previous project. The conflict resolution makes use of SDL validation concepts and is also based on pre-defined rules for resolving the conflicts.

The conflict resolution consists of two parts, resolving conflicts in the composition area when composing services, and resolving conflicts for profiles activated by the Calendar and Map functions simultaneously. Figure 4-33 shows an overview figure of the conflict resolution:

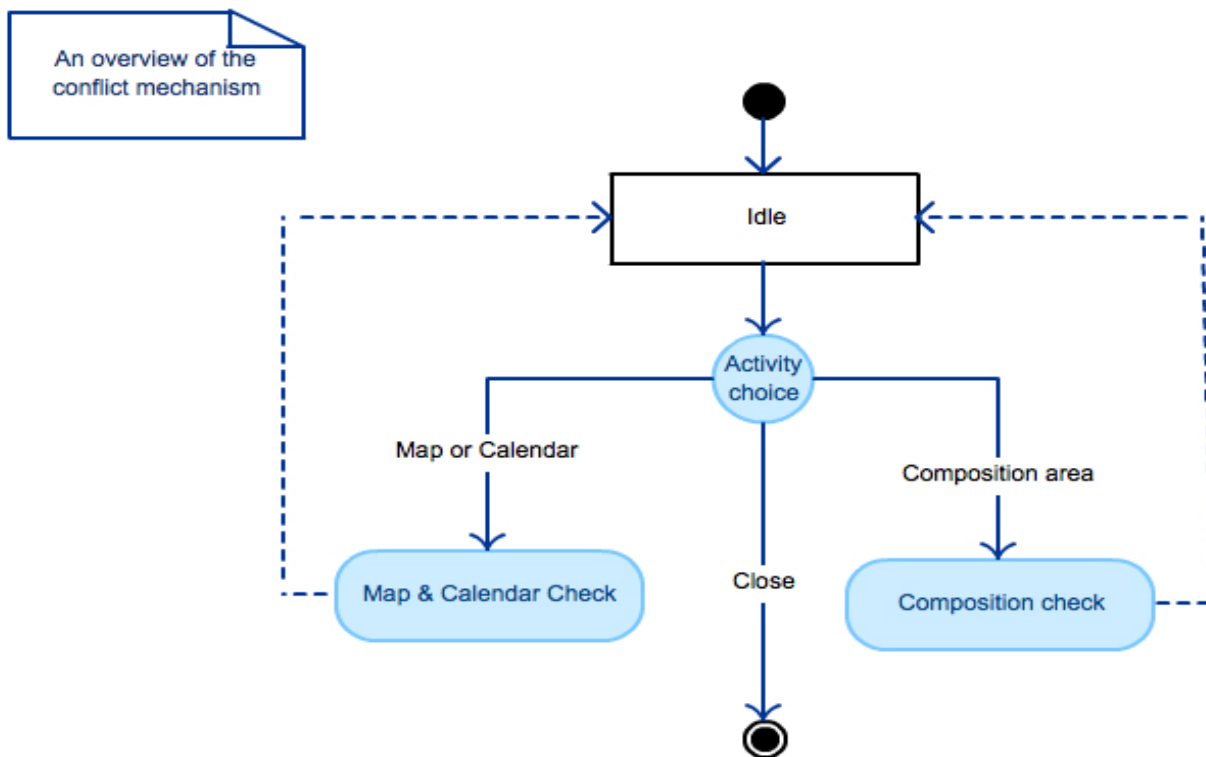


Figure 4-33: Conflict Resolution Activity Diagram

In the new system the Conflict Resolution has been further developed and integrated into the system.

As seen in Figure 4-33 the conflict resolution consists of two parts, resolving conflicts in the composition area when composing services, and resolving conflicts for profiles activated by the Calendar and Map functions simultaneously. From the figure we can see that there are two conflict activities checks: The Composition check and the Map & Calendar Check. The Composition Check is activated in the Profiles functionality when composing profiles in the GWT GUI. The Map & Calendar Check resides in the EasyDroid Engine. The conflict resolution is in the *Idle* state when none of these functionalities are activated. The Conflict resolution ends when the user closes the application.

Figure 4-34 shows an overview of the Composition check by a state diagram with activities:

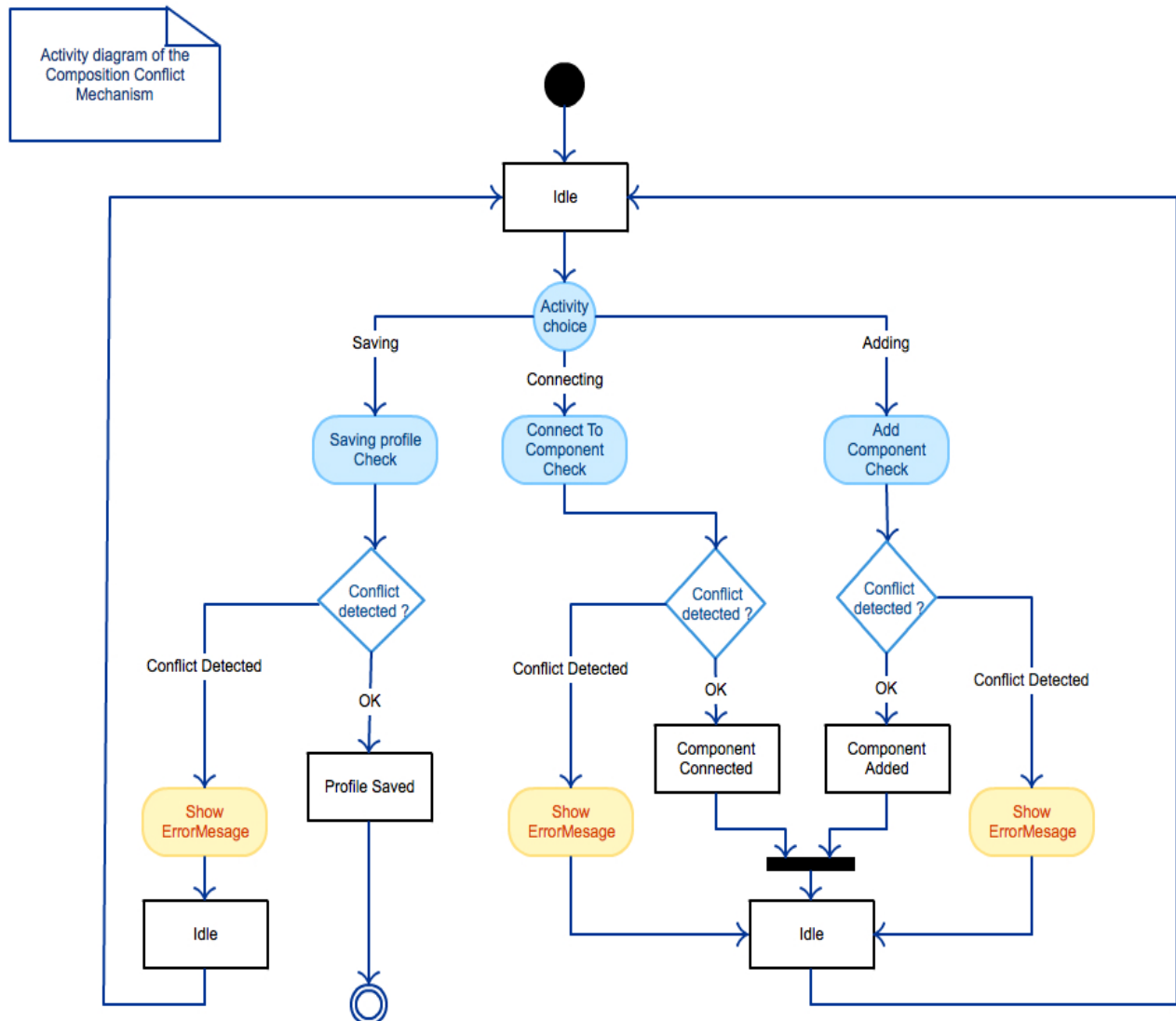


Figure 4-34: Composition Conflict Resolution Activity diagram

The Composition check for the Profiles functionality in the GWT GUI has the same activities as the Composition check for the Easy Composer application. However, the implementation of the activities is different.

From Figure 4-34 we can see that the conflict resolution for compositions has four states and three activities. The state diagram is described below:

- The initial state is the *Idle* state. When no interaction with the system is detected, the Idle state is active.
- Depending on the chosen activity, the *Activity Choice* navigates to the correspondent check activity.
- In the case when the *Add Component* or *Connect To Component* checks are activated and no conflict are detected, the component are added or connected, and it goes back to the initial *Idle* state. If a conflict is detected it displays a conflict-detected message and goes straight back to the initial *Idle* state.
- When a user tries to save a composition to a profile the *Saving Profile* check is activated. If no conflicts are detected, the composition is saved as a profile and the composition conflict resolution is complete. If a conflict is detected, a conflict-detected message will be displayed and it goes back to the idle state.

Furthermore we will describe each of the three checks activities in the diagram in Figure 4-34 in order to get a more in depth understanding of the conflict resolution. The check activities are annotated A for *Add Component Check*, B for the *Connect Component Check* and C for the *Save Profile Check*. The composition area consists of the following components: Incoming call, Caller List, Disconnection, Notification, Ringing and SendSMS.

A. Add Component Check: This activity checks if a component can be added to the current composition. The checks is be based on the following rule:

1. *Multiple instances:* There can be multiple instances of all components except the Incoming call component that can only have one instance.

The corresponding check to this rule is trivial, as it only needs to check if the incoming call component is already in the composition area or not.

B. Connect Component Check: This activity checks if a component can be connected to another component. The checks is based on the following rules:

1. *Already Connected:* The component should not be connected if the component already has a component connected to it.
2. *Compatibility:* The component connecting to the other component needs to be compatible with the component. This refers to Violation of (user specified) Correctness assertions [17] user design error explained in section 2.3.

3. *Livelock*: The connection should not create a loop that will cause a live lock in the composition. This refers to the Livelock condition explained for validating SDL in section 2.3 [16].

The following conflict-check resolutions are proposed for the rules above. The checks will be executed in a sequential manner with the ordering below:

1. *Already Connected*: The corresponding check for the first rule is quite simple, as it checks if the component already has a connection to it. If the component is already connected it will not be possible to connect to it.
2. *Compatibility*: For the second rule check we need to know which components are compatible with each other. To check for connection compatibility, a compatibility matrix has is presented in Figure 4-35 below:

The Connection
Compatibility Matrix

	IC	CL	Di	No	Ri	Sm
IC	N	Y	Y	Y	Y	Y
CL	N	Y	Y	Y	Y	Y
Di	N	Y	N	Y	N	Y
No	N	Y	Y	Y	Y	Y
Ri	N	Y	N	Y	N	Y
Sm	N	Y	Y	Y	Y	Y

Abbreviations:

IC - Incoming Call
CL - Caller List
Di - Disconnection
No - Notification
Ri - Ringing
Sm - Send SMS

Y - Yes
N - No

Figure 4-35: Component Compatibility Matrix

Here we have proposed a compatibility matrix that decides which components that can be connected to each other. The initiating component trying to connect to a component is represented in the first column (blue). The component that is being connected to is represented in the first row (red). If two components are compatible, this is denoted with the character “Y” for yes, if not it’s denoted with the character “N”. Note that the ordering of components is essential. For example, an Incoming Call can be connected to a Caller List, but not the opposite. If a conflict is detected the components will not be connected.

3. *Livelock avoidance*: Within the composition we will have a tables of the instances connected. If we connect the last element of this table to the first element we will create a

loop. The check will therefore check for this when a component tries to connect with another component. If a conflict is detected, it will not be connected.

C. Save Composition Check: This activity checks the composition for conflicts and if the composition can be saved as a profile. The checks is based on the following rules:

1. *Unconnected Components:* The composition should not contain unconnected components except for instances of the Notification and SendSMS components. This rule is referring to the Termination criteria [16], Liveliness criteria [16], Incompleteness criteria [17] and the Unreachable Components criteria [17].
2. *Deadlock:* The composition should not contain an ending component that causes a deadlock. This rule is referring to the Deadlock condition [16] explained in 2.3
3. *Non-functional composition:* The connection should not create known non-functional ordering of components. This rule refers Violation of (user specified) Temporal Logic formula [17] user design error explained in section 2.3.

The following conflict-check resolutions are proposed for the rules above. The checks will be executed in a sequential manner with the ordering below:

1. *Unconnected Components:* This check has slightly different checks for the different components. For the Ringing and Disconnect components, we check if they have an incoming connection and for the Caller List and Incoming Call components we check if they have an outgoing connection. The Notification and SendSMS component does not require a connection and can function as stand-alone components.
2. *Deadlock:* The only component that could cause a deadlock as a final state is the Caller List. The Deadlock check resolution will check if the Caller List is the final component.
3. *Non-functional composition:* Based on the compatibility matrix in Figure 4-35 we can have numerous of different combinations with only one instance of the component. Many of these combinations do not make sense or provide a real functionality as a profile. With more instances of components the number of combinations increases. Therefore it is cumbersome and slow to check for known illegal orderings. However there might be known non-functional orderings that occurs more commonly than others. One way to deal with this is creating a table of the known non-functional orderings in order to detect them. To collect information to the table we can have test users composing services and collect profile data from them.

Adding more components to the composition area can create new conflicts, and the conflict-check resolution needs to be updated. However, the same basic rules will apply. In the new column-tree version of the composer (small screen Profiles) presented in section 4.2.1, the combinations of compositions are already set. Hence, in this way the users can't compose the profiles wrong.

The Composition check for the Profiles functionality in the GWT GUI has the same activities as the proposed Composition check for the Easy Composer application. However, the implementation of the activities is different. The current implementation, only check the compositions when a Profile is saved. This is a simplified version of the conflict resolution proposed here. The implemented conflict resolution checks for multiple instances, compatibility and unconnected components. For the compatibility check it uses the compatibility matrix.

This is a combination of the three proposed conflict checks, but is only activated when a composition is saved. In the future, the conflict resolution should be further developed for the GWT GUI. The conflict resolution should be more similar to the tree proposed conflict checks. This conflict resolution should resolve livelocks, deadlocks and some non-functional compositions as well.

A new type of conflict occurs when a user reaches a location where a profile is set, and another profile is currently activated by time. Resolving this type of conflict in the GWT GUI is infeasible, and is resolved in the EasyDroid Engine at runtime. In the EasyDroid Engine we have created a temporary simple solution, which prioritize the profile that is set to the Calendar, as this is often a one-time occurrence of more importance.

In the future a more sophisticated conflict resolution will be designed and implemented for the EasyDroid Engine. A solution could be checking if the profiles functionalities actually are conflicting, based on the type of profile. A priority should be set for each profile based on the type. There should be a set of pre-defined types and priorities that the end-user can choose from. If two profiles are activated simultaneously, and is not the same type, they can both run without creating problems. If the profiles are of the same type, the profile with the highest priority should be the one running. By adding the type and priority attributes to each composition and having the conflict check resolution in the EasyDroid Engine, this type of conflict would be resolved.

In this chapter the design and implementation of the new system developed has been presented. The next chapter describes the testing carried out on the previous and new system. Furthermore, the results of the testing are presented.

5. System Test & Results

This chapter describes the tests and results during the Master Thesis period. The testing was carried out in the initial phase of the thesis period and at the end of the period. We did system and usability testing to carry out the testing of the tool. This points to the latest phase in Figure 2-1, where user testing is done to check if the developed solution is sufficient in terms of the user requirements. If the solution is insufficient a new iteration of the design process begins. The usability testing has been carried out using focus groups and field tests with feedback. The feedback is a pre-defined questionnaire [Appendix B], which every test participant fills out. Before the usability testing was carried out, we tested the system for errors and bugs, as well as doing a cognitive walkthrough [27] of the system. A test plan with different tasks was set up and we checked that these tasks were fulfilled without any problems. The next two sections will describe the system and usability tests.

5.1 System Testing

Before doing usability testing with real users, we needed to test the system by ourselves. For testing a system without end users, a cognitive walkthrough of the system is a good option. A cognitive walkthrough of a system is a method for checking usability issues in a system, focusing on the completion of user tasks. In other words, it is a formalized way of imagining people's reaction and actions to a user interface, using it for the first time. For a cognitive walkthrough it is important to have a predefined list of tasks descriptions and the actions it takes to accomplish each task [27]. For testing the new system we set up a list list of user tasks:

Task Description	Task Actions	Motivation
Log in to the system.	Type in username and password, press log in.	Check if the Log In mechanism works.
Make a new profile and save it in the GUI.	Add the components Incoming Call, Disconnect and SendSMS components to composition area. Fill in the SMS text. Connect the Incoming Call to the Disconnect component, and the Disconnect to the SendSMS component. Press Save and name the profile Busy.	Check that a simple profile is composed correctly and saved.
Activate the profile on a time and date.	Create a new event in the Calendar and add the profile Busy to a time in the near future.	Check that the new profile can be activated on an event.
Activate the profile on a location.	Create a new location in the Map and add the profile Busy to a location nearby.	Check that the new profile can be activated on a location.
Check if the profile is activated on the location.	Navigate to the specified location with the Android Device and have another test participant call the phone.	Check that the changes are synchronized with the Android Device and that the profile is acti-

		vated on the location.
Check if the profile is activated on the time.	Wait for the specified time period and have another participant call the Android Device.	Check that the changes are synchronized with the Android Device and that the profile is activated on the time.
Delete the Location from the Map.	Go to the Map, press the locations marker and then press the Remove button.	Check that a location is deleted.
Check that the profile is not activated on the location.	Navigate to the set location with the Android Device and have another test participant call the phone.	Check that the changes are synchronized with the Android Device and that the profile is not activated on the location.
Activate the Fast Activation functionality on a location.	Create a new location nearby the users location. Fill in input data for the SendSMS, Notification and WebConnect functionalities. Press save.	Check that the Fast Activation can be activated on a location
Check if the Fast Activation is activated on a location.	Navigate to the set location with the Android Device.	Check that the changes are synchronized with the Android Device and that the Fast Activation is activated.
Deactivate the Fast Activation on the location.	Press the location marker. Uncheck the Fast Activation boxes. Press Save.	Check that the Fast Activation can be deactivated on a location
Check that the Fast Activation is not activated on the location.	Navigate to the set location with the Android Device.	Check that the changes are synchronized with the Android Device and that the Fast Activation is deactivated.
Two users add each other to their Group and check the member location.	Each user adds a new user to the to the Group. Each user updates his location to the server in the Map by pressing the Update location button. Then the users navigate back to the Groups functionality and press the Refresh button. Finally, the users check each other locations by pressing the Option button and the Member Location button.	Check that the Buddy List System works.
Activate the Member Activation functionality on a member location.	Press the option data for the member added to the group. Fill in input data for the SendSMS and Notification. Press save.	Check that the Member Activation can be activated on a member location.

Check if the Member Activation is activated on a member location.	Make sure both have updated locations within each other radius. Check if the SMS is sent and that the notification pops up.	Check that the changes are synchronized with the Android Device and that the Member Activation is activated.
Deactivate the Member Activation on the member location.	Press the option data for the member added to the group. Uncheck the boxes for the SendSMS and Notification. Press save.	Check that the Member Activation can be deactivated on a member location.
Check that the Member Activation is not activated on the member location.	Make sure both have updated locations within each other radius. Check that no SMS is sent and no Notification pops up.	Check that the changes are synchronized with the Android Device and that the Member Activation is deactivated.

Table 5-1: Cognitive Walkthrough Task Description.

These tasks are focused on the main functionality. There are also less important functionalities that have been tested but were not included in the cognitive walkthrough. During the cognitive walkthrough, we asked ourselves these questions from [27]:

- ” 1. Will users be trying to produce whatever effect the action has?
 2. Will users see the control (button, menu, switch etc.) for the action
 3. Once users find the control, will they recognize that it produces the effect they want?
 4. After the action is taken, will users understand the feedback they get, so they can go on to the next action with confidence?” [27]

The first question is about what the user is thinking, as this is often different than what the designer is thinking. The user might do a different action than what the designer intended. The second question is about the users ability to locate control objects (buttons, menus etc.), meaning that the user simply notices the existence of the object. This is a more often problem than expected, and many users don’t notice control objects without help. The third question is connected to the second has to do with the user realizing and identifying what the control object purpose is. The final question is about the user understanding the feedback after the action is performed. In general, the system should give some sort of feedback depending of the action’s importance.

The cognitive walkthrough of the system enabled us to get results quickly. The majority of the tests went successfully. However, the tests revealed some problems with the updating of the user interface. For instance, if a new location was created in the Map GUI, this location did not appear instantly in the locations list. The Map GUI had to be refreshed for the new location to appear, and this was not very user friendly. In addition, the cognitive walkthrough helped us doing minor changes to the user interface that made it better. For instance, the control objects were made more obvious by using text as well as icons. The feedback in the Map and Groups functionality was also improved by having the active buttons highlighted with green for the correspondent buttons as seen in Figure 4-6 for the *draggable* marker.

5.2 Usability Testing

In the previous section the testing was carried out without the interaction of external test users. In this section the testing is focused on involving the test users. Initially in the Master Thesis period, the usability testing was carried out by doing field studies of the previous system. Before this period, usability-tests with focus groups had been carried out in order to improve the existing tool. In the end of this project phase the same kind test with a focus group was carried out to evaluate the new system. Both the field test and focus groups tests with results are described in this section.

5.2.1 Field test

For the field usability-test a group of seven people with different backgrounds tested out the previous system in the initial phase of the Master Thesis period. The test users ranged from professional experts and data students with technical backgrounds to social workers and sport athletes with less technical backgrounds. Hence, the field test represented novice to more experienced users. The purpose of the test was to see the potential for improvement in the previous system and also if users would actually make use of the tool. By having a field test we could get real test users instead of making our own assumptions of the limitation and improvements of the tool.

Before the actual field test where carried out, a short presentation of the tool was held. During the presentation, the tools functionality was briefly described and a simple scenario with tasks was carried out on the existing tool. Finally, the tool was installed on the test-users phones and the field study period began. The test-users used the tool for two weeks and the questionnaire [Appendix B] was filled out for the feedback. The questionnaires were then handed in and the results was analysed. The questionnaire [Appendix B] has 15 statements where the user can choose the level of agreement in an interval from 1 to 5, where 1 is the lowest and 5 is the highest level of agreement. The 15 statements are:

- 1) I believe I will use this system often.
- 2) I think the system was unnecessarily complicated.
- 3) I think the system was easy to use.
- 4) I believe I would need some help from a technical expert to be able to use the system.
- 5) I think the various parts of the system are well related to each other.
- 6) I think there was a lot of inconsistency in the system.
- 7) I believe most people can learn how to use this system very fast..
- 8) I think the system was difficult to use.
- 9) I was quite confident when using the system.
- 10) I need to learn a lot before I will be able to use the system by my self.
- 11) I think it is a good idea to decide the handling of incoming calls based on calendar events and locations.
- 12) I prefer to handle my incoming calls in the traditional way, i.e. by deciding what to do with calls when my phone starts ringing.

- 13) I think the existing features to handle incoming calls in some of the smart phones are better to use than this system.
- 14) I believe It is intuitive to create profiles using a graphical tool.
- 15) I think representing phone functionality using icons is complex.

The questionnaire score results for the different persons are presented in the table below:

<i>State- ment nr.</i>	<i>Professio- nal</i>	<i>Data Stu- dent</i>	<i>Social Worker</i>	<i>Social Worker</i>	<i>Telenor Group</i>	<i>Sports Athelete</i>	<i>Physiolo- gist</i>
1	3	2	4	3	2	4	3
2	3	2	2	2	3	2	2
3	5	4	3	5	4	5	4
4	1	1	1	1	1	1	1
5	4	3	4	5	4	3	3
6	3	2	2	3	2	3	2
7	3	3	4	5	2	4	4
8	1	2	1	1	3	1	2
9	5	1	4	2	4	5	3
10	1	2	1	1	2	1	1
11	5	3	5	5	3	5	5
12	4	3	1	2	5	3	4
13	3	2	1	1	3	3	2
14	5	5	3	5	3	4	4
15	1	1	2	1	4	1	1

Table 5-2: Questionnaire, Test Results

The score results of the questionnaire for the test users are presented in Table 5-2. The average score for each statement for the seven people is plotted in the column diagram below:

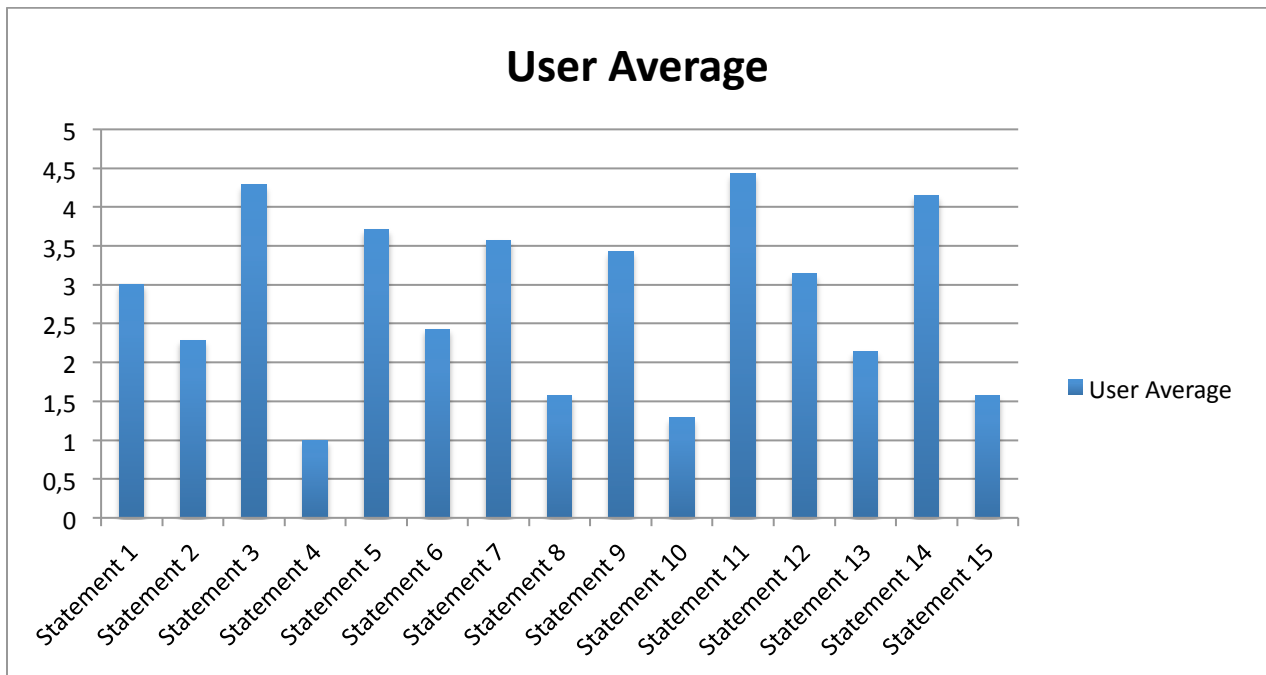


Figure 5-3: Column diagram, Questionnaire Average Score

Each column in Figure 5-3 corresponds to the average of the same fifteen statements from the seven questionnaires. The results of each statement is discussed below:

- 1) *I believe I will use this system often:* The average user was willing to use this system now and then. There was a potential for making the tool more attractive to use more often.
- 2) *I think the system was unnecessarily complicated:* The average user seemed that the system was not made unnecessarily complicated. However, there was potential for making it simpler.
- 3) *I think the system was easy to use:* The average user thought that the system was fairly easy to use. Furthermore, as we can see from Table 5-2, there were no significant variances between the users with a technical and non-technical background.
- 4) *I believe I would need some help from a technical expert to be able to use the system:* All of the users did not need help from a technical expert to use the system. This means that the previous system was fairly user-friendly. However, not having the need for a technical expert does not make the system sufficient in terms of user-friendliness.
- 5) *I think the various parts of the system are well related to each other:* The average user thought that that the features of the previous system elements was related to each other. This means that none of the functionality was unnecessary
- 6) *I think there was a lot of inconsistency in the system:* This results gives an indication that the previous system had some inconsistency. In other words the system lacked some consistency.

- 7) *I believe most people can learn how to use this system very fast:* This result showed us that the system had a fairly low time-to-learn. This is very important for the usability of the system.
- 8) *I think the system was difficult to use:* This is the same type of statement as number 3. The result was the same. The average user thought that the system was fairly easy to use.
- 9) *I was quite confident when using the system:* The result showed that the users were somewhat confident when using the system. However, by looking at Table 5-2 there is a big variance between the users. This means that the previous system had some potential for making the users more confident. It could also mean that the presentation of the system was not good enough, such that some users felt uncertain how the system worked.
- 10) *I need to learn a lot before I will be able to use the system by my self:* This is somewhat similar to statement 7. The results are consistent in that the system had a short learning time.
- 11) *I think it is a good idea to decide the handling of incoming calls based on calendar events and locations:* The test users think the concept of the previous tool was a good idea.
- 12) *I prefer to handle my incoming calls in the traditional way, i.e. by deciding what to do with calls when my phone starts ringing:* The users seem to be unsure if they want to handle their calls in the traditional way. This could be improved by making the users more confident in the system.
- 13) *I think the existing features to handle incoming calls in some of the smart phones are better to use than this system:* The average of the users thought the previous system solution was better than the existing solutions. This showed that there was a point in developing it further.
- 14) *I believe It is intuitive to create profiles using a graphical tool:* The users thought that composing profiles in the previous system is intuitive. This meant that the composing of profiles could stay the same.
- 15) *I think representing phone functionality using icons is complex:* The users thought that using icons representing the phone functionality is intuitive. This is connected to statements 14, and strengthened our assumption that the composing of profiles was intuitive.

Above the results of the statements of the questionnaire is presented. The test user could also give feedback commenting the usability and usefulness of the system. The main points are described below:

- The Idea and concept was widely accepted/acknowledged.
- The system was useful for families to monitor children, people travelling etc.
- The system has unique features compared to existing devices.
- It is better to only use a single interface/device than the previous system two interfaces.
- The system should do automatic downloading and updating.

- The users want a one-click solution to control their calls.
- The users think there are too many steps to get things done and that it takes too much effort and doesn't leave enough flexibility.
- Various profiles are needed but it must be possible to change the active profile from the device.
- Integration with Outlook or Google calendar is a must.
- Things should happen in the background without people noticing much.

The field test results implied that the previous system was widely accepted, but the need for having a single interface was significant. Furthermore, the system lacked some features in terms of usability and utility. This created a good starting point for developing the new system. In the next section the on-going user evaluation of the service composition tool with a focus group is presented.

5.2.2 Focus Group

The Focus Group is a on-going evaluation process where the development of the service composition tool is presented to a group of researchers and experts in the *UbiCompForAll* project. In the end of the project phase the new system was presented to this focus group. The people in this group had experience with the existing tool from the previous field test and earlier focus groups sessions. The purpose of the focus group was to present the main functionality to the group members and to get feedback if the new GUI was accepted.

The focus group accepted the new GUI, and the feedback resulted in some minor changes to the GUI. Furthermore, the focus group users wanted the EasyDroid Engine to be more independent of the GUI. This means that changes a user perform in the GUI on a device without the EasyDroid Engine would have immediate effect on the device with the EasyDroid Engine. Finally, the focus group wanted the GUI to be operational in offline mode on the smart phone.

6. Discussion & Future Work

In this chapter a discussion of the work carried out in the Master Thesis is presented. The discussion focuses on the limitations and improvements in the new system compared to the previous system as well as some other aspects. Finally, the remaining and future work on the service composition tool is described.

6.1 Limitations & Improvements

The new system has improved from the previous system in terms of usability and utility. Before the new GUI was developed, research was carried out on design principles and guidelines in order to make the GUI with adequate quality in terms of usability. Paper prototyping was also used to test the user interface before implementing it. With the paper prototyping we were able to discover design faults early in the project phase.

Having a common single user interface is an improvement by itself. The time for a new user to get familiar with the new system decreases drastically. This is important for the new user to accept the new system. If it takes too long to learn the system, the user will most likely not use it. The new GUI is web based, hence it can be accessed from every device that has a web browser. In the previous system this was not possible. The user had to install the application (Easy Composer) on the computer. With the new system the user only need to install the Android part of the system. The web-based solution will make future development of versions for other devices than Android much simpler, since the developer does not need to take the GUI into consideration. The user is also able to do changes on-the-fly when using the smart phone. In the previous system this functionality was limited and created frustration with the user not having control of what's going on. Also, the synchronisation part of the two GUIs is now eliminated. However, synchronization has to take place between the web server and the Android Engine. The limitation of having the GUI web-based is that the user needs to be online to access it.

The new system also has a number of improvements in the different functionalities. The Calendar and Profiles functionalities are quite similar to the previous system. The profiles are created the same way, and the adding of events in the Calendars is done in the same manner. However, the GUI is adapting to the screen size of the device. Adapting to the screen size and using an appropriate GUI makes it easier for the user to carry out the tasks. Furthermore, a new way of creating profiles has been developed. This is the new column-tree version of the composer (small screen Profiles) presented in section 4.2.1.2. Composing profiles in the traditional way has proven to be cumbersome for smart phone screen sizes. The new column-tree version of the composer is an intuitive way to make profiles on the fly. In addition, it is impossible to create wrongly composed profiles in this way. It is also faster and easier for new users. The limitation of this approach is that not all the possible profiles in the larger screen version can be made with the tree-column version. Also, adding new components to the tree-column version, creates many new possibilities, which means more work for the developer. While the Profiles and Calendars functionalities has not improved too significantly the Maps and Groups functionalities has undergone substantial improvements in terms of usability and utility.

First of all the response time of the Maps functionality has improved drastically from the rather slow previous version. In the previous system the user had limited possibility for adding a location in the Maps functionality. The user could only search for addresses in order to create a new location. This creates problems when the user does not know the address of the location or if the new location does not have an address. This is solved by having a *dragable* marker, which can be dragged to the exact position in the map area. The user can now use the search feature, as well as the *dragable* marker. Furthermore, the user now has the option of switching between a map, satellite and hybrid view in the Google Maps controls. This could be useful if the user want to search for specific buildings or places that does not show in the map, but the satellite view. The user also has the option whether the location markers should be showed in the map area or not with the Overlays feature. In addition the Overlays feature interacts with some of the Groups functionality by enabling the user to show the group members location. Another improvement is that the user can show the device location in the map area if the application is running on an Android Device with the EasyDroid Engine installed. This location can then be uploaded to server, such that other users can retrieve this location. This is a part of the Buddy List System, which is integrated in the new system.

Adding a profile to a location is done in the same way in the new system as for the previous system. In addition, the new Map functionality also has the feature Fast Activation added to a location. The Fast Activation helps the user to set up a text message, notification or/and open a web site quickly for the locations. The text message and notification features could be achieved in the previous system by making profiles with one component. However, this would take much longer time and effort by the user. The user also has the option of connecting to web sites, which creates a number of new possibilities of utility with the tool. The user can now customize each location with it's own web site that would pop up. Although many improvements have been made to the Maps functionality, there are some limitations. One limitation of the new Maps functionality is that the user has to press the location marker in order to add a profile to the location. This might not be intuitive for first time users, and can be made more intuitive by highlighting the markers or some other sort of feedback to the user. Another limitation is that users might not remember that the fast activation or profile is activated for a location. A solution for this is changing the colour of the location markers to a different colour when either Fast Activation or a profile is activated.

The Groups functionality was added to the previous system in the previous project period. In the new system the Groups functionality has been improved in terms of usability and utility. With the new grid approach showed in Figure 4-13, a record in the grid can be edited by simply pressing the field the user wants to edit. This is especially helpful if the user wants to change the member's group. With the previous system the user had to remove and add the member in order to do the change the member fields. In addition, the grid approach allows the user to sort the grid list records on name, groups and also choose between a grouped and ungrouped view.

The most significant improvement with the Groups functionality is that a location is attached to each group member. This is a part of the Buddy List system developed for the previous system, but never integrated. Obtaining members location raises privacy and security issues, and therefore a authentication method was made in order to make sure the users are authenticated before the member location can be retrieved. If a user adds one member to one of his/her groups means that the user give permissions to the added member for checking the users location. This authen-

tication method has its limitations. The fact that the user needs to remove the member from his groups in order to hide the users location is cumbersome. Also the users should get a notification that another user has added them to their group. Another limitation in the new Groups functionality is that the user has to manually press the refresh button in order to receive the member's newest location updates. This functionality will eventually reside in the EasyDroid Engine where the newest members location updates will be periodically fetched from the database directly.

With the location being attached to a group member, the concept of the Member Activation was developed. The idea of being notified or automatically send a text message when specified members is within or without the users location adds new utility for the Groups functionality. For instance, this could come in handy for families with small children or old demented people that can wander off not knowing where their whereabouts.

The two main limitation of the new system is the fact that the synchronization between the EasyDroid Engine and the database goes through the GWT GUI and that the GWT GUI cannot be used offline. The synchronisation does not create a problem when the GWT GUI is used on a device with the EasyDroid Engine installed. The problem arises when a user does changes in the GWT GUI on a computer, and the changes are not updated in the EasyDroid Engine. These changes will not be updated before the GWT GUI is loaded on the phone. Making the EasyDroid Engine more independent of the GWT GUI can solve this problem. The GWT GUI should also be available offline. This can be achieved by using HTML5 caching. Solving these issued will be described further in the future work.

The user-centered design process involved users in an early stage of the development process, which allowed the developers to receive feedback before the initial implementation of the system. This enabled the project participants to detect design errors and changes before implementation, which potentially saved us much time and effort. The field tests carried out in the project has resulted in informative feedback that we have used for developing the new system. The field tests gave us both quantitative and qualitative feedback. However, the field tests should have more test persons in order to get more realistic results. For testing the new system we did a cognitive walkthrough without test users. When the conflict resolution is further developed in the future, this should be tested as well in the cognitive walkthrough.

The new system has not been tested thoroughly enough with real life users. Furthermore, the Buddy List system with the Member Activation and the Fast Activation needs to be tested in more real environments with real users. The same kind of field tests carried out in the early phase of this project period should be carried out for the new system. The reason that this has not been done is due to the fact that we did not have time or resources for this kind of testing at the particular time. The focus was on getting the main functionality of the system as stable as possible, and present this to the focus group.

6.2 Future Development

The new system has improved substantially from the previous system during this project period. However, there is some remaining work to be carried out before the system is ready for the public market. Based on the work in the project, the following can make the tool better in terms of usability and utility:

- First of all the EasyDroid Engine should do synchronization with the database independently of the GWT GUI. This can be achieved by having the EasyDroid Engine periodically request the database for the latest updates. The requests can then be handled by a web service that fetches the Composition, Fast Activation, and Member Activation XML and sends it back as a response.
- The GWT GUI should also be operational in offline mode. This can be achieved in GWT using HTML5 with caching. The browser will cache the compiled GWT files and make them available offline.
- The Conflict resolution should be further developed for the GWT GUI and the EasyDroid Engine. Furthermore, it should be tested with a cognitive walkthrough.
- The new system should be tested thoroughly with field tests, in order to get user feedback on the new functionality begin added. In addition, the Buddy List system should be tested out with a special field test with a number of users coordinating.
- The new system should also support other platforms than Android. There should also be versions of the Engine ported to iPhone and WinPhone. This would increase the potential number of users drastically.
- The Fast Activation can be integrated in the Calendar functionality. This would expand the Fast Activation utility context, being able to activate on time as well as locations. Also a choice for disconnecting a call should be incorporated into the Fast Activation. This can be realised by having a check box for disconnecting a call. This would make the number of steps fewer for the end user.
- The Calendar functionality should interact with the Google or Outlook calendars, such that the activities of the external calendars are showed in the Calendars functionality.

7. Conclusion

In this Master Thesis a tool for end-user service composition has been studied and further developed. During the thesis period a new system was developed based on the functionality of the previous system. Furthermore, new functionality has been added to the system. Parts of the new functionality were based on concepts from my previous project and parts of it were based on concepts developed in this project period. The main task carried out has been to create a single new graphical user interface (GUI), which functions as a common interface for all devices. In addition, the EasyDroid Engine was remade and a new server side of the system was created. Furthermore, a background study of literature and work related to the different topics of the objectives, has been performed in order to achieve a sufficient quality of the work to be done. Finally, usability testing has been carried out on both the previous system using field tests and the new system making use of a cognitive walkthrough.

Initially in this project period field tests were carried out in order to identify the potential for improvements in the previous system and also if users would actually make use of the tool. After the field test period, the feedback was analysed and a meeting with the project participants was held. The meeting resulted in a project plan where a new GUI was to be developed using GWT. This resulted in a background literature study on user-friendly design methods and getting familiar with developing using GWT. After the background study, the user requirements were specified and a paper prototype created. This paper prototype was then evaluated in terms of the user requirements. This corresponds to one iteration in a user-centered design process. Two iterations were carried out before the implementation of the GUI began. The field tests, background study, and utilizing a user-centered design process with paper prototyping has contributed to understand the user needs and requirements at an early stage and also improving the system with a sufficient quality in terms of utility and usability.

The actual implementation of the GUI begun after the paper prototypes was approved and the project plan was set up. The development of the GUI was in accordance with common design principles and guidelines learned in the background study. During the development phase, meetings were held every second week to see if the development was on schedule with the original project plan. The development carried out resulted in a user-friendly GUI in GWT with a client and server side using remote procedure calls for communication. The newly developed system only lacked communication with the EasyDroid Engine of the previous system. Using this EasyDroid Engine turned out to be cumbersome, as the user interface and the engine part was strongly connected. Therefore, the EasyDroid Engine was remade without the GUI and only the components that were significant for the new system. Creating the EasyDroid Engine by ourselves made it easier to add new functionality, such as the Fast and Member Activation functionalities. However, having to remake the EasyDroid Engine created a significant delay in the project schedule.

The new system has improved considerably since the previous system. The usability of the system has improved with the fact that the user only needs to relate to a single web based GUI. Furthermore, the GUI adapts to the screen size of the device. This enables the user to do changes to the system from anywhere with any device that has an Internet connection. Also the functionalities of the tool have improved in terms of usability, specially the Map and Groups functionalities.

The Map functionality has a draggable marker feature that enables a user to create a location anywhere on the map area. The user can also display the device location in the map area. In addition, the user can choose which overlays and map views that should be displayed in the map area. The Groups functionality has a more structured view using a grid with editable entries. In addition, the grid allows the user to sort on name, groups and also choose between a grouped and ungrouped view.

Furthermore, the utility of the system has also improved considerably. The Fast Activation enables the user to do send text messages and receive notifications on different locations in a simple and fast manner. Also, the new Web Connect opens up a new world of possibilities by allowing the user to show content from the Internet on locations. In addition the Buddy List system allows the user to share its own location, and also display other users updated locations. Furthermore, the Member Activation attaches the SendSMS and Notification components of Fast Activation to the member's location. This creates new possibilities for the user, as the member's location is dynamically as opposed to the static locations in the Map functionality. A new way of composing profiles has also been developed for small screen devices by using tree-structured columns.

There is still some remaining work before the public market can use the system. The two most important features is the synchronization between the EasyDroid Engine and the server database, and that the GWT GUI works in offline mode. In addition, the conflict resolution should be developed further. Furthermore, field tests should be carried out to receive real test user feedback on the new system. When the EasyDroid Engine is published, versions of the EasyDroid Engine for other platforms should be developed, in order to create a larger user base. The service composition tool has evolved within this project period in terms of usability and utility. With the huge increase of Smart Phone users in the recent years the service composition tool will most likely benefit many end-users having the need of creating their own services.

Appendix

Appendix A: References

1. Weiser, Mark, "Ubiquitous computing", www.ubiq.com/hypertext/weiser/UbiCompHotTopics.html, August 1993
2. M.P. Papazoglou, D. Georgakopoulos, "Service-Oriented Computing", Communications of the ACM – Service-oriented computing, Volume 46 Issue 10, October 2003.
3. Papazoglou. M.P., Traverso, P., Dustdar, S., Leymann, F. : "Service-Oriented Computing: State of the Art and Research Challenges," Computer, vol.40, no.11, pp.38-45, Nov.2007
4. Wikipedia foundation: User-Centered design, http://en.wikipedia.org/wiki/User-centered_design, last modified 23. November 2011
5. Usman Wajid, Abdallah Namoune, and Nikolay Mehandjiev. 2010. "A comparison of three service composition approaches for end users". In *Proceedings of the International Conference on Advanced Visual Interfaces (AVI '10)*
6. Sintef, UbiCompForAll, <http://www.sintef.no/Projectweb/UbiCompForAll/>, accessed May 2012
7. Floch, J., Herrmann, P., Kahn, M.U., Sanders, R., Stav, E., Sætre, R.: "End-User Service Composition in Mobile Pervasive Environments", 2011
8. Blomkvist, S. (2002). Persona – an overview Personas and goal-directed design. *Retrieved November, 22, 1-8, September 2002*
9. Gaffney, G. Scenarios, "What is a *Scenario*?" (2000) <http://www.infodesign.com.au/ftp/Scenarios.pdf>, 2000
10. Schneiderman, Ben, Plaisant, Catherine, Designing the User Interface, 4th Edition, 2004
11. Snyder, Carolyn, Paper Prototyping, <http://www.cim.mcgill.ca/~jer/courses/hci/ref/snyder.pdf> 2001, Accessed February 2012
12. Hazas, M.; Scott, J.; Krumm, J.; "Location-aware computing comes of age," *Computer*, vol.37, no.2, pp. 95- 97, Feb 2004
13. Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. 2010., "Improving energy efficiency of location sensing on smartphones.", In *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*
14. Google Foundation, Google Latitude, <http://www.google.com/mobile/latitude/>, accessed May 2012

15. Andrew Tennyson, Google Latitude Information, http://www.ehow.com/info_8110096_google-latitude-information.html, accessed November 2011
16. Werner, C., S. Kraatz, et al. (2006). A UML Profile for Communicating Systems System Analysis and Modeling: Language Profiles. R. Gotzhein and R. Reed, Springer Berlin Heidelberg. 4320: 1-18. 2006
17. Gerard J. Holzmann. 1992. "Practical methods for the formal validation of SDL specifications". *Comput. Commun.* 15, 2 March 1992
18. Google Developers, Google Web ToolKit, Overview, <https://developers.google.com/web-toolkit/overview>, accessed February 2012
19. Google Developers, Google Web ToolKit, Server Communication, <https://developers.google.com/web-toolkit/doc/1.6/DevGuideServerCommunication>, accessed March 2012
20. Google Developers, Google Web ToolKit, JSNI , <https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsJSNI>, accessed March 201
21. Mbaabu, Frank, "Tool-chain development for end-user composite services", Master-Thesis, August 2011
22. Android Developers, WebView, <http://developer.Android.com/reference/Android/webkit/WebView.html>, accessed May 2012
23. Edwin A. Hernandez, "War of the Mobile Browsers," IEEE Pervasive Computing, pp. 82-85, January-March, 2009
24. Sintef, UbiCompForAll Scenarios, <http://www.sintef.no/Projectweb/UbiCompForAll/Results/Scenarios/>, accessed May 2012
25. Jackson, Joab, Oracle pulls the plug on old JavaFX runtime, <http://www.javaworld.com/javaworld/jw-02-2012/120228-java-fx-script-deprecated.html>, accessed March 2012
26. Android Developers, <http://developer.Android.com/reference/Android/telephony/TelephonyManager.html>, accessed April 2012
27. Lewis, Clayton, Rieman, John, Task+Centered User Interface Design, Chapter 4 Evaluating the Design Without Users, <http://hcibib.org/tcuid/chap-4.html#4-1>, accessed May 2012

Appendix B: Questionnaire

Some questions about the system you have used

Please set 'x' in only one choice per question

	Strongly Disagree					Strongly Agree
1) I believe I will use this system often	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
2) I think the system was unnecessarily complicated	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
3) I think the system was easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
4) I believe I would need some help from a technical expert to be able to use the system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
5) I think the various parts of the system are well related to each other	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
6) I think there was a lot of inconsistency in the system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
7) I believe most people can learn how to use this system very fast	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
8) I think the system was difficult to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
9) I was quite confident when using the system	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	
10) I need to learn a lot before I will be able to use the system by my self	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	5	

- 11) I think it is a good idea to decide the handling of incoming calls based on calendar events and locations
- 12) I prefer to handle my incoming calls in the traditional way, i.e. by deciding what to do with calls when my phone starts ringing
- 13) I think the existing features to handle incoming calls in some of the smart phones are better to use than this system
- 14) I believe It is intuitive to create profiles using a graphical tool
- 15) I think representing phone functionality using icons is complex

Strongly Disagree Strongly Agree

1	2	3	4	5

1	2	3	4	5

1	2	3	4	5

1	2	3	4	5

1	2	3	4	5

Please include any comments regarding the usability and usefulness of the system.



Appendix C: XML representations

C1. Exported Composition XML

```
<Compositions>

<Composition id="1" name="test" type="1" firstCid="21349265999507" De-
scription="Enter Description"><Settings ringerVolume="50.0" alarm-
Volume="50.0" mediaVolume="50.0" brightnessLevel="50.0" stateType="2"
GPSon="1" WIFIon="1" MobileDataOn="1" />

<Criteria>
</Criteria>

<Component posX="580" posY="111" isMovable="true" specificsNotSpeci-
fied="true" cid="21349265999507" name="IncomingCall" ali-
as="IncomingCall" type="-1" nextCid="21349266020019" nextCidI-
fMatch="0" nextCidIfNoMatch="0" matchType="1" action="0"><Connector
cid="21349265999507" cbid="1" nextCid="21349266020019"
/></Component><Component posX="831" posY="138" isMovable="true" spe-
cificsNotSpecified="true" cid="21349266020019" name="Connect" ali-
as="Connect" type="2" nextCid="0" nextCidIfMatch="0" nextCidI-
fNoMatch="0" matchType="1" action="4"><Connector cid="21349266020019"
cbid="1" nextCid="0" />
</Component>

</Composition>

<Composition id="2" name="Available" type="1"
firstCid="21349265999507" Description="Enter Description"><Settings
ringerVolume="50.0" alarmVolume="50.0" mediaVolume="50.0" brightness-
Level="50.0" stateType="2" GPSon="1" WIFIon="1" MobileDataOn="1" />

<Criteria>
<Criteria id="104" name="Shoi" description="ad">
<Time id="104" name="Shoi" compositionId="2" fromtime="20120430 1632"
totime="20120430 1732" />
</Criteria>

<Criteria id="233" name="Nidaros3">
<Location id="233" name="Nidaros3" compositionId="2" address="" lon-
gitude="10.39178" latitude="63.43017" radius="1" activate="Within
radius" />
</Criteria>

<Criteria id="234" name="NTNU2">
<Location id="234" name="NTNU2" compositionId="2" address="" longi-
tude="10.412592887878418" latitude="63.414827850393685" radius=
"1011" activate="Within radius" />
</Criteria>

</Criteria><Component posX="580" posY="111" isMovable="true" specif-
icsNotSpecified="true" cid="21349265999507" name="IncomingCall" ali-
```

```
as="IncomingCall" type="-1" nextCid="21349266020019" nextCidIfMatch="0" nextCidIfNoMatch="0" matchType="1" action="0"><Connector cid="21349265999507" cbid="1" nextCid="21349266020019" /></Component><Component posx="831" posy="138" isMovable="true" specificsNotSpecified="true" cid="21349266020019" name="Connect" alias="Connect" type="2" nextCid="0" nextCidIfMatch="0" nextCidIfNoMatch="0" matchType="1" action="4"><Connector cid="21349266020019" cbid="1" nextCid="0" />
```

```
</Component>  
</Composition>
```

```
<Composition id="5" name="Busy" type="1" firstCid="21367210581411" Description="Enter Description"><Settings ringerVolume="50.0" alarmVolume="50.0" mediaVolume="50.0" brightnessLevel="50.0" stateType="2" GPSON="1" WIFION="1" MobileDataOn="1" /> <Criteria>
```

```
<Criteria id="105" name="Lecture" description="TTM 4410">  
<Time id="105" name="Lecture" compositionId="5" fromtime="20120611 1300" totime="20120611 1400" />  
</Criteria>
```

```
<Criteria id="249" name="Tyholt">  
<Location id="249" name="Tyholt" compositionId="5" address="" longitude="10.437312126159668" latitude="63.42195231840607" radius="500" activate="Within radius" />  
</Criteria>  
</Criteria>
```

```
<Component posx="402" posy="101" isMovable="true" specificsNotSpecified="true" cid="21367210581411" name="IncomingCall" alias="IncomingCall" type="-1" nextCid="21367210624003" nextCidIfMatch="-1" nextCidIfNoMatch="-1" matchType="1" action="0"><Connector cid="21367210581411" cbid="1" nextCid="21367210624003" /></Component><Component posx="635" posy="137" isMovable="true" specificsNotSpecified="true" cid="21367210624003" name="Disconnect" alias="Disconnect" type="2" nextCid="-1" nextCidIfMatch="-1" nextCidIfNoMatch="-1" matchType="1" action="1"><Connector cid="21367210624003" cbid="1" nextCid="-1" />
```

```
</Component>  
</Composition>
```

```
<Compositions>
```

C2. Fast Activation XML example

```
<Fastactivations>
  <Fastactivation>
    <LoacationID>14</LoacationID>
    <Longitude>"10.437312126159668"</Longitude>
    <Latitude>"63.42195231840607"</Latitude>
    <Radius>"500"</Radius>
    <Nactivated>"true"</Nactivated>
    <Sactivated>"true"</Sactivated>
    <Wactivated>"true"</Wactivated>
    <NTicker>"Hello"</NTicker>
    <NTitle>"Remember"</NTitle>
    <NMessage>"Remember lecture"</NMessage>
    <SNumber>"92822836"</SNumber>
    <SMessage>"Hi, I'm at Campus"</SMessage>
    <WURL>"http://www.ntnu.no"</WURL>
  </Fastactivation>
</Fastactivations>
```

C2. Member Activation XML example

```
<Memberactivations>
  <Memberactivation>
    <MemberID>34</MemberID>
    <Longitude>"10.237312126159668"</Longitude>
    <Latitude>"63.72195231840607"</Latitude>
    <Radius>"200"</Radius>
    <Nactivated>"true"</Nactivated>
    <Sactivated>"true"</Sactivated>
    <NTicker>"Hello"</NTicker>
    <NTitle>"Nearby"</NTitle>
    <NMessage>"Frank is nearby"</NMessage>
    <SNumber>"92822836"</SNumber>
    <SMessage>"Hi, I'm within a 200 m radius of your loca-
tion"</SMessage>
  </Memberactivation>
</Memberactivations>
```